Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Automatisering og elektronikkdesign	Vårsemesteret 2023 Åpen
Forfattere: Simen Sande, Simon Lundgren & Eirik Tyvold	
Fagansvarlig: Morten Mossige Veiledere: Morten Mossige & Yngve Finnestad	
Tittel på bacheloroppgaven: Måling av lakktykkelse Engelsk tittel: Measurement of laqcuer thickness	
Studiepoeng: 20	
Emneord: Robot, Instrumentering, Automatisering, Programmering, Seriell kommunikasjon, GUI	Sidetall: 56 + vedlegg/annet: 88 Stavanger 15. mai 2023

Forord

Denne bacheloroppgaven fullfører bachelorgraden for automatisering og elektronikkdesign ved Universitetet i Stavanger, og er skrevet av Simen Sande, Simon Lundgren og Eirik Tyvold. Rapporten er skrevet våren 2023 med veiledning av Morten Mossige og Yngve Finnestad, som begge er ansatt i ABB, avdeling Bryne.

Valg av bacheloroppgave er basert på felles interesse for å kunne bruke teorien tilegnet gjennom studiene i en praktisk oppgave. Oppgaven har vært utfordrende, lærerik, og krevd mye arbeid på grunn av omfanget. Det som har vært spesielt interessant med denne oppgaven er å få utvikle et system fra start til slutt, med praktisk arbeid, prosjektering og programmering.

Vi vil takke Morten Mossige for veiledning og tilgjengelighet for spørsmål under oppgaven. En stor takk rettes også til Yngve Finnestad for praktisk og teknisk hjelp. Samtidig skal Jonas Rifsgård takkes for all hjelp med å finne materiell og opparbeiding av testplater. Vi vil også takke alle ved ABB avdeling Bryne for muligheten til å utføre bacheloroppgaven hos dere.

En spesiell takk rettes til Lasse Ånestad for gode diskusjoner, innspill, løsningsforslag og ikke minst forarbeid med kode for kommunikasjon mellom motordrivere og PC som ble tildelt ved oppstart av bacheloroppgaven.

Sammendrag

ABB avdeling Bryne er et utviklingssenter for å utvikle og forbedre lakkeringsroboter. Imidlertid har ikke senteret en pålitelig metode for å måle lakktykkelse. For tiden utføres målinger manuelt ved bruk av et håndholdt måleinstrument, som kan resultere i upresise resultater. Det er derfor behov for en automatisert måleprosess som kan øke nøyaktigheten og effektiviteten til målingene. Denne bacheloroppgaven har som mål å utvikle et slikt automatisert system for å løse dette problemet. Med et slikt system vil det være mulig å utføre større tester med langt flere målinger og med høyere presisjon og nøyaktighet. Det vil derfor også være mulig å sammenligne målinger over samme område på ulike tester. Dette vil kunne bidra til å forbedre kvalitetskontrollen og øke produktiviteten i lakkeringsprosessen.

I denne bacheloroppgaven har det blitt utviklet et kartesisk robotsystem med tre akser og tilhørende programkode for å styre systemet. Ved å kombinere mekanisk design, elektronikk og programmering har en funksjonell prototype som er i stand til å utføre en rekke bevegelser med høy presisjon blitt utviklet. En viktig del av oppgaven var å utvikle programkoden som styrer systemet. Dette involverte å utvikle algoritmer for å konvertere ønsket bevegelse til en serie av motorstyringskommandoer. Det ble også implementert en sikkerhetsfunksjon for å sikre at systemet opererte innenfor trygge grenser og ikke forårsaket skade på omgivelsene eller brukeren.

Systemets arbeidsområde består av et tre akse bord på omtrent én kvadratmeter, hvor en sensor forflyttes over arbeidsområde for å ta målinger. Systemet skal kunne ta imot et valgfritt antall målinger over et brukerdefinert område. Antall målinger og måleområde blir definert ved oppstart av lakkeringsprogrammet. Tanken er at en skal kunne ta testplater som er ferdig lakkert, plassere de i arbeidsområdet, og bruke systemet til å kvalitetsteste lakkeringen. Arbeidsoppgavene i prosjektet består av praktisk arbeid og utvikling av programvare. Det praktiske arbeidet består av montering av tre akse bordet, samt kabling og montering av tilhørende utstyr som stegmotorer, motorkontrollere og annet nødvendig materiell. Programmeringen går hovedsakelig ut på å lage et program som skal bestemme hvordan systemet kjører, basert på ønsket resultat fra brukeren.

Hver akse i systemet forflyttes ved bruk av stegmotorer. Pulssignaler fra motorkontrolleren styrer pådraget til stegmotorene. Motorkontrollerene kommuniserer med en PC for å motta kommandoer ved bruk av en seriell kommunikasjonsprotokoll. For å utføre målingene brukes et måleinstrument som kobles til PC-en via et annet serielt grensesnitt. Dette gjør det mulig å kontrollere systemet på en effektiv måte fra en sentral datamaskin. Det er også implementert ulike metoder for å fremvise resultatene av lakkmålingene. Dataen kan presenteres i ulike tredimensjonale plott samt et todimensjonalt. En mulighet for å lagre og anvende data er også tilgjengelig ved at data blir lagret med tilhørende koordinatsystem. Dette gjøres for å ha muligheten til å presentere målingene på andre måter enn tilrettelagt i programmet.

Innhold

Kapittel 1

Introduksjon

Dette kapittelet innleder bacheloroppgaven, som tar utgangspunkt i å lage et automatisert system for måling av lakktykkelse ved hjelp av en sensor. Kapittelet skal gi en forståelse for hvorfor dette er relevant og interessant for både samfunnet og ABB, som er bedriften oppgaven utføres for.

1.1 Innledning

Automatisering er et uttrykk som først ble introdusert etter første verdenskrig, men metoden ble tatt i bruk mye tidligere. Det var spesielt under den industrielle revolusjonen, som startet på slutten av 1700-tallet, at automatiserte systemer ble tatt i bruk. I starten var dette hovedsakelig mekaniske systemer hvor menneskelig arbeidskraft ble erstattet med maskiner. I nyere tid er automatisering ofte programmer som utføres sekvensielt, eller som regulerer et pådragsorgan [3] .

Hensikten med å automatisere arbeidsoppgaver er å unngå å måtte gjøre repeterende arbeid manuelt. Måten slike automatiserte systemer blir utviklet er ved bruk av algoritmer som blir programmert og implementert i utstyr som kan utføre arbeidsoppgavene. Dette er spesielt effektivt for arbeidsoppgaver som ikke endrer seg over tid eller som endres ved forutsette mønstre. Dersom et system er avhengig av å endres ved bruk av vurdering og tolkning, blir det straks mye vanskeligere, og en må konstruere et mye mer kompleks system.

Ettersom behovet for å automatisere prosesser både i hverdagen og i arbeidslivet øker, er det interessant å nevne noen fordeler. I hverdagen vil en fordel av å automatisere arbeidsoppgaver være å kunne gi enhver mer tid til å fokusere på mer trivielle opplevelser, som kan resultere i økt livskvalitet. Automatisering av arbeidsoppgaver i arbeidslivet, vil kunne åpne for at arbeidskraft kan benytte kompetansen sin på andre områder. Dette kan potensielt øke kapasiteten, arbeidskraften og lønnsomheten i bedriften.

Det er derfor svært spennende å kunne få utvikle et automatisert system til en av arbeidsoppgavene i lakkeringsprosessen til ABB. Selve bacheloroppgaven er gitt av ABB, avdeling Bryne, og tar utgangspunkt i måling av lakktykkelse. ABB Bryne er et av tre globale utviklingssentre for utvikling av roboter i ABB. Avdelingen utvikler ny teknologi og nye produkter for energisektoren. De spesialiserer seg på automatisering av lakkering, og har en del av ansvaret for utvikling av ABB sine lakkeringsroboter [2]. Verdens første

lakkeringsrobot ble utviklet i Trallefrabrikken på Bryne, og står på utstilling i resepsjonen til ABB avdeling Bryne [1], se figur 1.1.

Figur 1.1: Verdens første lakkeringsrobot

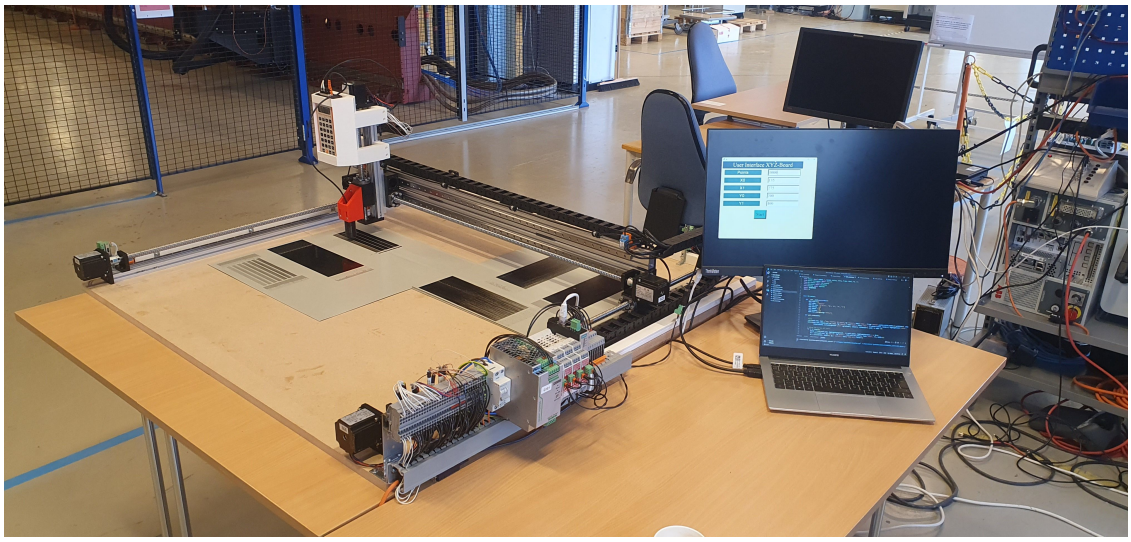
Lakkering er ikke bare estetikk, men har også muligens en viktigere funksjon ved å kunne beskytte mot slitasje påført av eksterne krefter. Slik slitasje kan komme i mange former, noen eksempler kan være regn, støv eller snø. Over tid vil slik slitasje føre til korrosjon, som er en faktor til at en opplever for eksempel rust på biler. Lakkering vil kunne redusere denne korrosjonsprosessen, altså være med å øke levetiden til materialet som lakkeres [14].

Ved å øke levetiden til ulikt material som er viktig i samfunnet vil en skape verdiøkning, ettersom en reduserer behovet for utbytting av slitte deler. Slike deler kan for eksempel bestå av forskjellige typer plast benyttet i bilproduksjon. Plast er hovedsakelig produsert av petroleumsbaserte stoffer, altså råolje og naturgass [26] [19]. Utvinning og bruk av råolje og naturgasser er store bidragsytere til klimagassutslipp. Dermed kan det tenkes at lakkering av utstyr er med på å skape et bærekraftig samfunn, samtidig som det bidrar til arbeidet med det grønne skiftet.

Selv om lakkering ofte er lønnsomt, er det en svært kostbar prosess. Det er derfor viktig at lakkeringsprosessen sørger for et jevnt lag med lakk over området som skal beskyttes, og at

det ikke brukes mer enn nødvendig. Dermed er det interessant å kunne verifisere tykkelsen på lakk over et gitt område. Dette er for å kunne si noe om kvaliteten på fordelingen av lakken. Det er nettopp et slikt system som skal konstrueres i denne bacheloroppgaven.

Formålet med oppgaven er å måle tykkelsen på lakk ved hjelp av et Elcometer. Elcometer er et apparat som kan måle tykkelsen på lakk ved bruk av en sensor. I denne oppgaven gjøres det ved at det blir laget et tre akse bord, med et arbeidsområde på om lag én kvadratmeter. Det skal plasseres en plate med lakk i arbeidsområdet og systemet skal skanne platen etter ønsket antall målinger og sende data i retur som forteller noe om tykkelsen på lakken. For å oppnå en samling av data som gir en god approksimasjon av lakkeringen, må det utvikles et system for å forflytte sensoren jevnt over arbeidsområdet. Forflytningen skjer ved bruk av stegmotorer som beveger sine respektive akser basert på pådrag gitt fra systemet. Figur 1.2 viser hvordan systemet ser ut.



Figur 1.2: Bilde viser systemet

For å kunne analysere data innhentet av systemet, må dataene presenteres og lagres. Det benyttes forskjellige fremvisninger for å kunne visualisere resultatene på ulike måter, som kan gjøre det enklere å analysere resultatene. En metode for å lagre datasettene som en excel fil er implementert, for å ta vare på dataen som er samlet inn. Det gjøre det også mulig å presentere lakkmålingene ved bruk av andre metoder dersom det skulle være behov ved et senere tidspunkt.

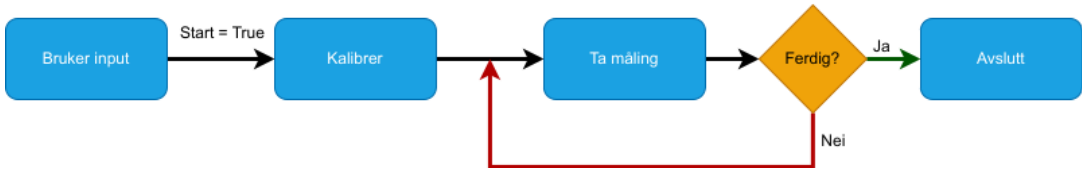
Bakgrunnen for et ønske om et automatisk målesystem for lakktykkelse på ABB Bryne, er at det i dag ikke er noen god metode for å måle lakktykkelse. Dersom en skal måle tykkelse på lakkeringen, må en fysisk bruke Elcometeret ved å manuelt utføre målingene. Denne metoden er både tidkrevende og upresis ettersom metoden og områdene som blir målt kan variere avhengig av personen som fysisk utfører målingene. Ved å automatisere denne prosessen vil en kunne få en nøyaktig og presis metode til å systematisk ta flere måleprøver av lakkplater som er sammenlignbar.

1.2 Beskrivelse av oppgaven

Arbeidsoppgaver:

- Bygge brettet.
- Trekke kabler.
- Koble alt relevant utstyr.
- Lage program for styring av motorer.
- Innhenting av data
- Behandle, lagre og presentere data.

Figur 1.3 viser et overordnet flytskjema av programstrukturen. De blå firkantene viser til en egen prosess som er detaljert beskrevet i kapittel 3. Rombene symboliserer et spørsmål hvor utfallet *ja*, eller *nei* påvirker videre utførelse av programmet.



Figur 1.3: Flytskjema viser en overordnet programflyt av systemet

Som vist i figuren 1.3 starter programmet etter at brukeren har trykket *Start*. Det vil kjøre frem til alle målingene er utført og deretter avslutte.

1.3 Oppbygging av oppgaven

Bacheloroppgaven er delt inn i seks hovedkapitler. Kapittel én introduserer oppgaven og forteller litt om hvorfor oppgaven er aktuell. Kapittel to introduserer bakgrunnen til hvilke utstyr og programmer som blir brukt i oppgaven, samt generell informasjon vedrørende lakkering. Kapittel tre omhandler hvordan systemet er satt sammen, og hvordan programmet er utviklet. Kapittel fire tar for seg resultatene av testene som blir utført i oppgaven. Avslutningsvis tar kapittel fem for seg drøfting og diskusjon vedrørende oppgaven, og kapittel seks viser konklusjonen.

1.4 Prosjektgruppe

Prosjektgruppen består av tre studenter ved Universitetet i Stavanger som studerer Automatisering og elektronikkdesign. To i gruppen har erfaring fra arbeidslivet og har fagbrev i elektrikerfaget, som tidligere ble kalt gruppe L.

Kapittel 2

Bakgrunn til oppgaven

Formålet med dette kapittelet er å gi en omfattende innføring i ulike teknikker innen lakkering og måling, samt å presentere utstyret og programvaren som vil bli anvendt i systemets design-, konstruksjons- og kjørefaser. Målet er å gi leseren en grundig forståelse av teknologiene som benyttes i oppgaven, samt å forklare hvordan de ulike komponentene og programvarene samarbeider i systemet.

2.1 Generell bakgrunnsinformasjon

Lakkering er tilføying av lakk på forskjellige flater eller objekter. Lakk er hovedsakelig en beskyttende film, som består av naturlig eller kunstig fremstilte harpikser. Lakk brukes for å hindre slitasje, men blir samtidig ofte brukt i sammenheng med estetikk. Det finnes mange bruksområder for lakk, noen eksempler kan være billakk eller båtlakk, hvor lakken sin funksjon er både estetisk og beskyttende [20].

Det finnes flere forskjellige metoder for å lakkere. En spesiell metode ofte benyttet i bilindustrien er elektroforetisk lakkering. Elektroforese er en prosess hvor protoner og elektroner beveges i et elektromagnetisk felt. Protoner dras mot den negative polen, samtidig som elektroner dras mot den positive polen [15]. Lakken som benyttes er ioniserte lakkharpikser, ofte med vann som løsemiddel. Prosessen består av et lakkeringsobjektet som blir spenningssatt og senket i lakkeringsbad som resulterer i at de ioniserte harpiksene samles rundt overflaten til lakkeringsobjektet. Dette er spesielt gunstig ved lakkering av gjenstander hvor det er vanskelig å komme til [25].

Spray Vision er en bedrift som spesialiserer seg på kvalitetssjekk av belegg. En av metodene de bruker for å analysere konvensjonell spraymaling er ved bruk av testlapper. Det brukes roboter til å spraymale ulike testlapper som kan legges i en *SprayCapture SprayGUN*, som er en type skanner som brukes til å generere digital visualisering av beleggene på testplatene [10]. Denne måten å analysere spraymaling ved bruk av testlapper er en inspirasjon til hvordan systemet i dette prosjektet kan gjøre noe tilsvarende ved måling av lakktykkelse.

Systemet som konstrueres i dette prosjektet vil kunne verifisere lakkeringtykkelsen ved bruk av testlapper. Et eksempel kan være dersom det skal verifiseres lakkering av karosseri på bil. Da må det festes flere testlapper rundt karosseriet. Etter lakkering av karosseriet kan en overføre disse testlappene til systemets arbeidsområde. Systemet kan skanne testlappene

og gi en god indikasjon på fordelingen til lakkeringen over karosseriet. Kvaliteten på testene er avhengig av hvor mange testlapper som blir festet rundt karosseriet, og hvor mange målinger som blir utført. Denne metoden kan tenkes å være gunstig til bruk på andre objekter også.

Disse lakkeringsprosessene er kostbare. Dersom store områder skal lakkeres er det viktig å sørge for at tilstrekkelig mengde lakk benyttes for å oppnå ønskelig beskyttende effekt, samt ønskelig levetid. Samtidig er det viktig å ikke bruke for mye lakk for å redusere kostnadene og ressursbehovet. For å kunne bestemme om kvaliteten av lakkering er god nok, og riktig mengde lakk er brukt, er gode metoder for å måle lakken nødvendig.

2.2 Komponenter og grensesnitt

I dette delkapittelet presenteres det ulike komponenter og kommunikasjonsmetoder som blir benyttet ved utvikling av den kartesiske roboten.

I denne bacheloroppgaven brukes det et måleinstrument fra Elcometer. Selve instrumentet er et håndholdt måleinstrument som er laget for å måle tykkelse på belegg. Elcometeret kan utstyres med forskjellige prober som bruker to ulike metoder for å måle beleggtykkelse. Dersom det skal måles belegg som ikke er magnetisk på magnetiske substrater som for eksempel stål, brukes prinsippet for elektromagnetisk induksjon. Den andre metoden er basert på prinsippet om *eddy current* som brukes dersom det skal måles belegg som ikke er konduktiv på substrat som ikke er magnetisk [8].

Figur 2.1: Elcometer

Lakkeringen presentert i denne oppgaven utføres på substrater av stål. Det betyr at proben som blir benyttet anvender prinsippet for elektromagnetisk induksjon. Prinsippet tar utgangspunkt i å produsere elektrisitet ved hjelp av magnetisme. Dette gjøres ved å bruke

spoler, altså viklinger av elektriske ledere som for eksempel kobber, for å generere strøm i en krets. Strømmen i kretsen genereres når det elektromagnetiske feltet gjennom spolen endres. Det er slik proben i denne oppgaven fungerer. Proben bruker et system med tre spoler, hvor den sentrale spolen er spenningsatt av Elcometeret. De to andre spolene er på hver sin side av den sentrale spolen, og detekterer endringen i det magnetiske feltet generert av den sentrale spolen. Når proben nærmer seg noe som endrer det magnetiske feltet vil balansen av det magnetiske feltet endres. Endringen gjør at det er mulig å måle spenningen mellom spolene som kan benyttes til å avgjøre avstanden til objektet [24].

Figur 2.2: Måleproben som kobles til Elcometeret

Elcometeret måler disse spenningsverdiene fra proben, og behandler dataen. Måleinstrumentet har en måleusikkerhet på $\pm 1\%$ eller $\pm 1 \mu m$, og har mulighet til å fremvise statistisk interessante verdier som standardavvik, gjennomsnittlig tykkelsesmåling, laveste måleverdi og antall målinger. Instrumentet kan lagre opp til 10 000 målinger, og har mulighet til å lagre de i 200 forskjellige *batcher*. Det er mulig å sende dataen fra Elcometeret til pcen, da denne har mulighet for RS-232 kommunikasjon [9].

RS-232 er en standard for seriell kommunikasjon som brukes til å overføre digital data mellom to enheter. Kommunikasjonen involverer to ledere, en sender og en mottaker, som er koblet sammen via en seriell kabel. Når data sendes, sendes hvert bit sekvensielt fra senderen, en etter en, til mottakeren. Hver bit i datasekvensen representeres av et spenningsnivå på ledningene, som kan være positiv eller negativ. En positiv spenning indikerer en logisk én, mens en negativ spenning indikerer en logisk null[6, s. 43-46].

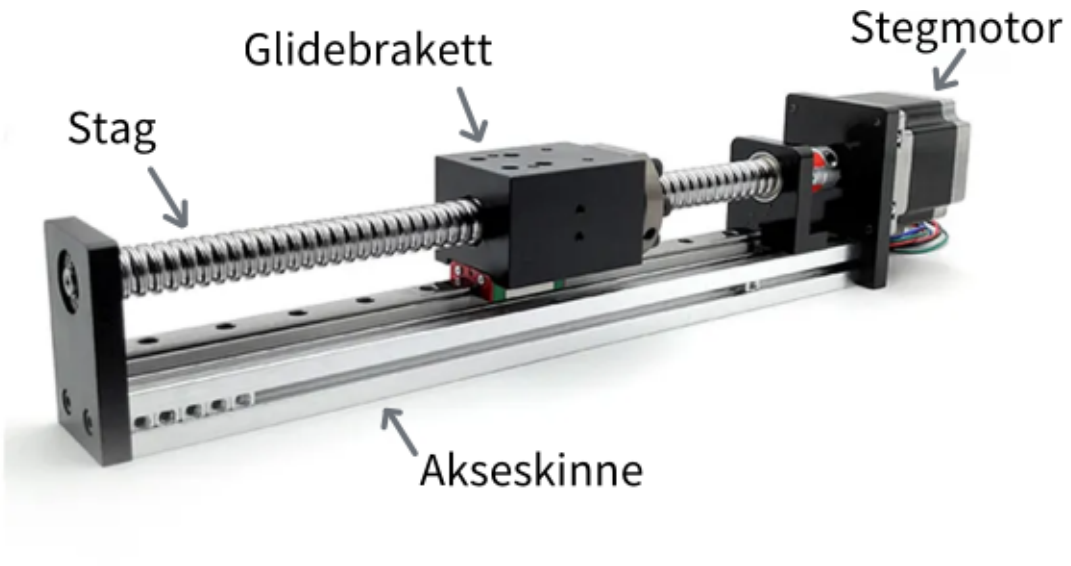
RS-232 standarden spesifiserer også parametere som hastighet, antall databit, stoppbit og paritet, som er nødvendige for å sikre riktig overføring av data. Hastigheten bestemmer hvor fort data overføres, mens antall databit bestemmer størrelsen på hvert dataord. Stoppbit brukes til å indikere slutten av hvert dataord, mens paritet brukes til å sjekke om dataene som er overført, er riktig. For å kommunisere via RS-232, må begge enhetene være konfigurert til å bruke samme hastighet, antall databit, stoppbit og paritet. Når data sendes, overvåker mottakeren spenningsnivået på ledningene for å rekonstruere datasekvensen. Hvis datasekvensen ikke tilsvarer det forventede mønsteret, kan en feil oppstå, og dataene må sendes på nytt. [6, s. 45]

I dag brukes RS-232 sjelden til å kommunisere mellom datamaskiner på grunn av begrensningene i hastighet og avstand. RS-232 brukes imidlertid fortsatt i mange andre applikasjoner, som industrielle automatiseringssystemer, medisinsk utstyr og konsollporter på datamaskiner.

For å bevege sensoren og Elcometeret rundt arbeidsområdet benyttes det fire stegmotorer. Stegmotorerene som brukes i oppgaven er børsteløse DC motorer, som omdanner elektrisitet til mekanisk energi. En typisk stegmotor er bygget opp av en stator med flere

viklinger og en rotor som er kontinuerlig magnetisk. Rotering genereres ved at viklingene blir spenningssatt ved pådrag fra en kontroller. Hvor fort motoren roterer og orientering av rotasjonen bestemmes av rekkefølgen og hurtigheten kontrolleren spenningssetter viklingene [4, s. 1].

UMot SW40L1000 10C7 er en lineær akse med tilhørende stegmotor som er brukt i denne oppgaven.

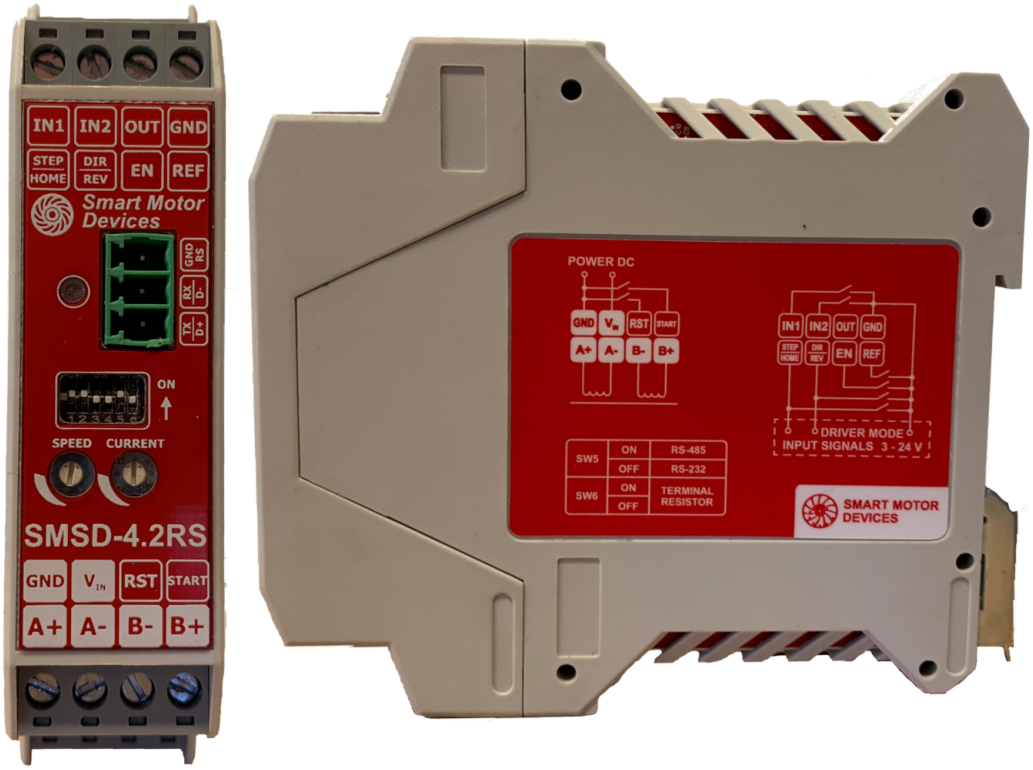


Figur 2.3: Hentet fra [28]

Aksen har en arbeidslengde på 1000 mm, gjengeavstand på 10 mm og skrue diameter på 16 mm. Stegmotorene driver en glidebrakett langs aksen, og har en nøyaktighet på 0.05 mm. Motorene kan maksimalt flytte glidebraketten med en kraft på 188 N og en hastighet på 250 mm/s uten last. Hver motor trekker maksimalt 2 A og har en steglengde på 1.8°, se vedlegg C.2. Hvor mange steg motoren bruker for en rotasjon kan vises matematisk

$$\frac{360^\circ}{1.8^\circ} = 200 \tag{2.1}$$

SMSD-4.2RS er motorkontrolleren som styrer pådraget til stegmotorene. Kontrolleren kan ta imot spenning mellom 12-48 V og er designet for å operere med stegmotorer med et maksimalt strømtrekk på 4.2 A. Kontrolleren styrer stegmotorene ved å sende pulser, med ulik hastighet og rekkefølge avhengig av formålet til kommandoen, på fire ledninger som er koblet til de ulike viklingene i stegmotoren. Det er også mulig å justere mikro steppingen til driveren. Valgmulighetene når det kommer til mikro stepping er $\frac{1}{1}, \frac{1}{2}, \frac{1}{4}$ eller $\frac{1}{16}$. Driveren har tre ulike kontrollmoduser. Programmerbar, analog hastighetskontroll og pulskontroll. I dette prosjektet blir driveren operert i programmerbar modus hvor det sendes kontinuerlig ASCII kommandoer fra pc-en når programmet kjøres [7].

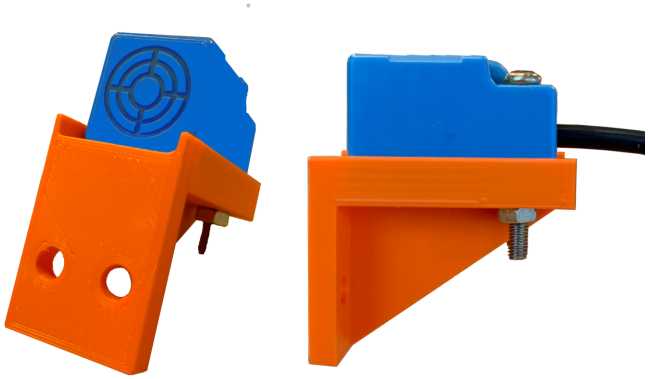


Figur 2.4: SMSD-4.2RS

Driveren kan benytte RS-232 og RS-485 kommunikasjon for mottak og sending av ASCII kommandoene. RS-485 er en standard for seriell kommunikasjon som brukes til å overføre digitale data over lengre avstander. Denne standarden definerer elektriske egenskaper, signalnivåer, signalhastighet og protokoller som brukes for å overføre data mellom flere enheter. RS-485 bruker en differensiell signaloverføring, som betyr at dataene overføres ved hjelp av to ledere, en positiv og en negativ. Dette gir bedre støyimmunitet og bidrar til å redusere elektromagnetisk interferens fra omgivelsene[6, s. 80-82].

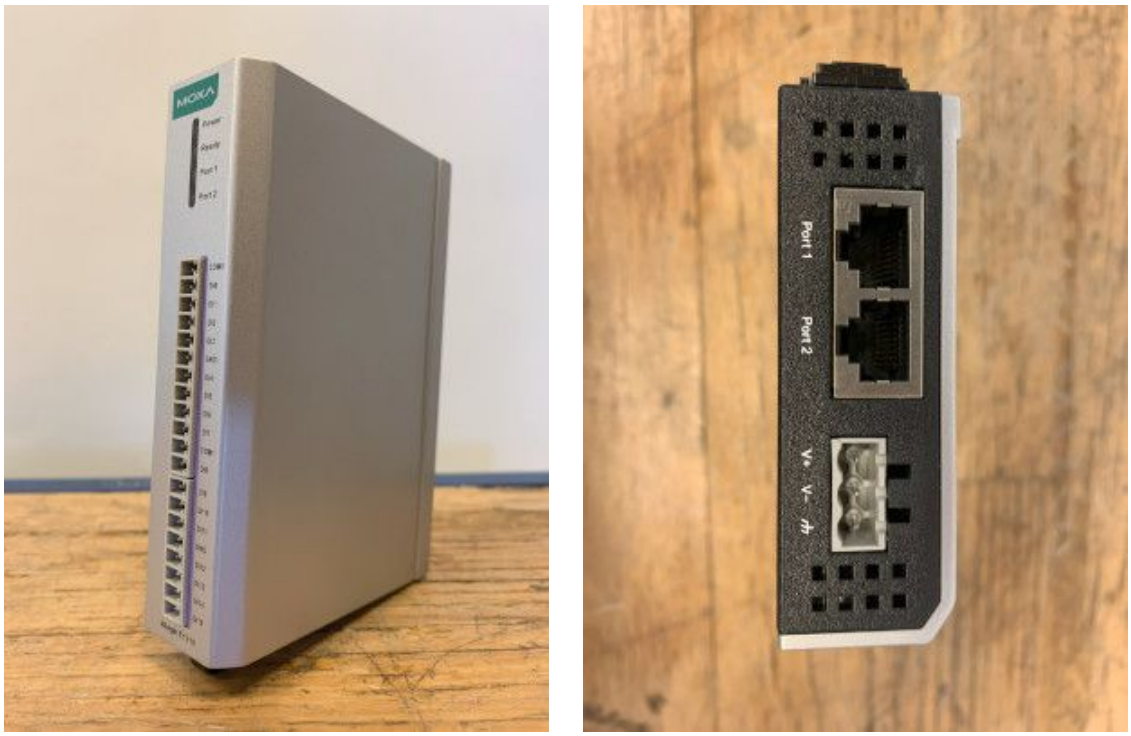
Vanligvis består RS-485 kommunikasjonsoppkoblingen av en master-enhet og flere slave-enheter. Master-enheten sender kommandoer til slave-enhetene, og hver slave-enhet svarer med data som er relevant for forespørselen. Alle enhetene bruker samme kommunikasjonslinje, og det er en nøye koordinert overføring av data. RS-485-standarden har en maksimal overføringshastighet på opptil 10 Mbps, men den vanligste hastigheten er 9600 bps. Protokollen som brukes for å overføre data, avhenger av applikasjonen og enhetene som brukes. RS-485 er en pålitelig og kostnadseffektiv måte å kommunisere serielt. Det brukes vanligvis i industrielle applikasjoner, inkludert fabrikker, bygninger og andre steder hvor det er behov for å overføre data over lengre avstander.[6, s. 79]

Som et sikkerhetstiltak er det implementert induktive sensorer i systemet. Induktive sensorer er ofte brukt til å registrere avstanden til objekter av metall. Deteksjonslengden til sensorene kan variere i henhold til jernholdigheten til objektet. Det vil si at metallobjekter som inneholder mye jern, detekteres på lengre avstand enn for eksempel et objekt av kobber eller aluminium.



Figur 2.5: SN04-N

En induktiv sensor består i prinsippet av en spole som har viklinger rundt en jernkjerne. Spolen er koblet sammen med en kondensator og til sammen genererer de et elektormagnetisk felt i nærheten av sensorens overflate. Når sensoren kommer innen rekkevidde av metall, blir den induserte spenningen endret, som resulterer i at en elektronisk bryter aktiveres [13, s. 32]. SN04-N er den induktive avstandsensoren montert som sikkerhetstiltak i dette systemet. Sensoren opererer på en spenning mellom 12-24V, og har en deteksjonslengde på 4 mm, men anbefalt avstand er 3.2 mm. Sensoren fungerer med en *normally open* bryter, slik at når den detekterer objekter så lukkes bryteren, og sender ut et signal på utgangen.



(a) Digitale innganger på Moxa

(b) Strømforsyning og signalutgang

Figur 2.6: Moxa ioLogik E1210

Signalet fra sensoren går inn på både motordriveren og på en Moxa iologik io-enhet. En moxa er en innputt/utputt enhet også kalt I/O-enhet, det vil si at den kan ta inn et signal fra en type grensesnitt og sende det videre på et annet grensesnitt. Dette gjør det mulig å hente informasjonen fra de induktive sensorene inn på pcen. I/O-enheter kan konfigureres på mange ulike måter, enten med digitale- og analoge innputter og utputter eller en blanding av disse. Enheten som blir brukt i dette systemet er en Moxa ioLogik E1210. Den er utstyrt med 16 digitale innputter. De digitale innputtene har en terskel verdi på 0-3 VDC for *av* tilstand og 10-30 VDC for *på* tilstand[17]. Et API (Application Programming Interface) blir brukt for å hente ut diverse parametre fra moxaen. API er et grensesnitt i en programvare som kan si til annen programvare at den skal kjøre spesifikke operasjoner [18].

2.3 Programmer benyttet i oppgaven

Dette delkapittelet tar for seg programmer som er essensiell for både design- og kjørefasen til systemet. Programmet til systemet er skrevet i python. Python er et høynivå, tolket programmeringsspråk som ble utviklet på slutten av 1980-tallet [11]. Det er kjent for å være et enkelt, intuitivt og fleksibelt programmeringsspråk som er populært blant utviklere for en rekke applikasjoner, inkludert webutvikling, datavitenskap, maskinlæring, spillutvikling og automatisering. Python støtter også flere måter å programmere på, inkludert funksjon-, objekt- og logikkorientert programmering [22].

I oppstartsfasen var det et behov for å sjekke kommunikasjon mellom enheter i systemet. En enkel måte å gjøre dette på var ved å bruke Tera Term, som er et gratis, åpent kommunikasjonsprogram som støtter flere forskjellige kommunikasjonsprotokoller. En av disse protokollene er seriell kommunikasjon, som blir benyttet til å både kommunisere med motordriverne og Elcometeret [29].

Ettersom braketter til ulike formål var nødvendig i konstruksjonsfasen, ble modelleringsprogrammet Tinkercad brukt. Programmet kan benyttes til å modellere og simulere elektriske kretser. Det er også mulig å generere kodeblokker, og lage modeller som kan 3D-printes [5].

Tilstandsdiagrammer og flytskjemaer som blir presentert i oppgaven er laget ved bruk av Draw.io. Det er et online diagramverktøy som gjør det mulig å lage flytskjemaer, diagrammer, tankekart og mye annet. Draw.io er en nettbasert applikasjon som er fullt integrert med Google Disk [16].

Det har vært behov for et program til å redigere bilder ettersom de fleste bildene presentert i oppgaven er bilder som er tatt med kamera, enten av systemet eller enheter når de er koblet fra systemet. Gimp er et gratis bildebehandlingsprogram som blir brukt til å behandle og presentere bildene i oppgaven [27].

R er et programmeringsspråk som fokuserer på statistiske beregninger og visualisering. R tilbyr en bred samling av statistiske teknikker, som for eksempel lineær og ulineær modellering, statistiske tester og tidsserieanalyser. En av R sine styrker og grunnen til at det blir brukt i denne oppgaven, er de brukervennlige høy kvalitets plotte funksjonene, som for eksempel *Box plot* [12].

Kapittel 3

Design og konstruksjon av maskin- og programvare

3.1 Maskinvare

I dette delkapittelet blir design- og konstruksjonsfasen av roboten presentert. Kapittelet tar for seg fremgangsmåten vedrørende praktisk arbeid samt ulik prosjektering av arbeid knyttet til oppgaven.

3.1.1 Innledning

For å lage en robot som kan utføre ønsket oppgave om å automatisere en prosess som kan ta målinger av et lakkert objekt, blir det konstruert en 3-akse robot, også kjent som et kartesisk robotsystem. Systemet inneholder tre kartesiske akser, X Y Z som vil kunne bevege et verktøy langs de tre aksene ved hjelp av fire forskjellige stegmotorer. Dette er roboter som ofte brukes i industrielle pakking og plasseringsoperasjoner. Disse robotene kan brukes til enkle flytteoperasjoner, hvor en ønsker at et objekt skal flyttes i en retning og plasseres på samme sted hver gang.

3.1.2 Montering av en kartesisk robot

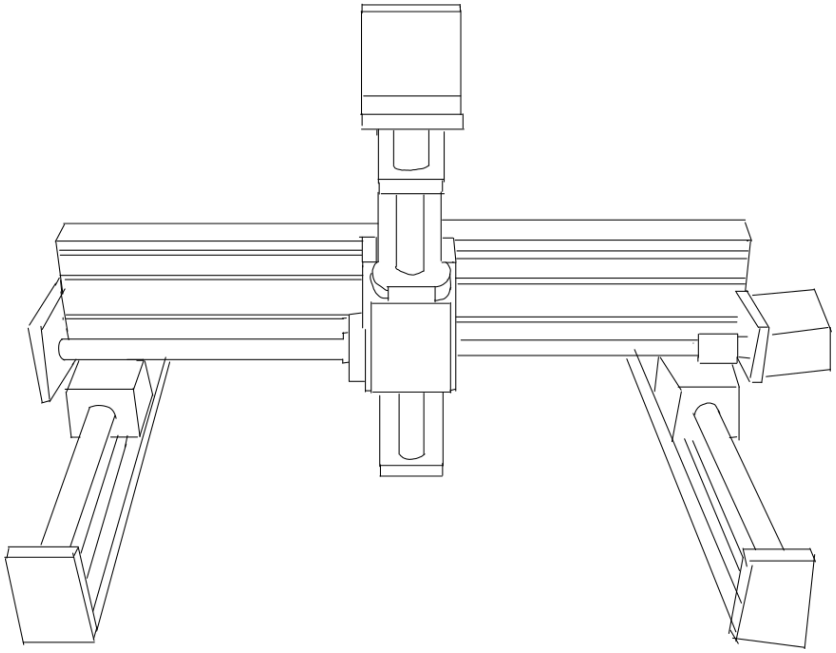
For konstruksjon av selve roboten har ABB bestilt nødvendige deler. Det ble ved oppstart utlevert en eske med diverse innhold som blir brukt i konstruksjonsfasen.

Figur 3.1: Materiell til oppgaven

Esken inneholdt følgende deler.

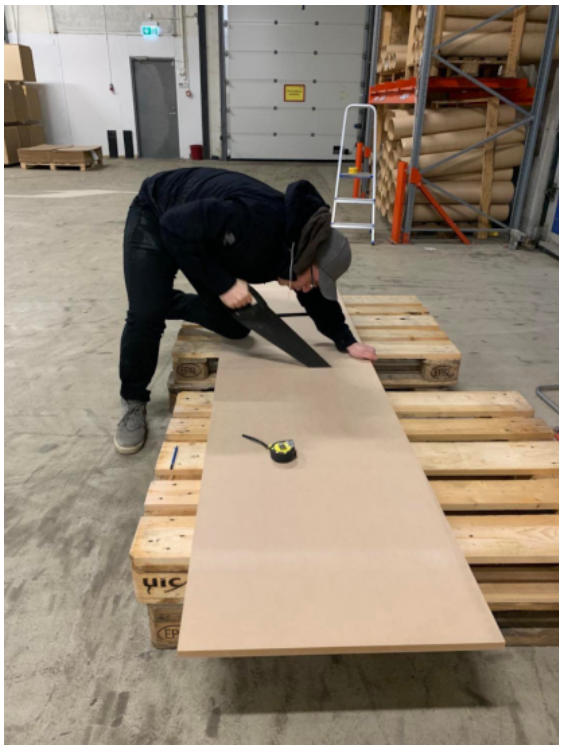
- Umot stegmotor med tilhørende akse på 1000 mm (SW40L1000 10C7) x3
- Umot stegmotor med tilhørende akse på 300 mm x1
- Motorkontroller (SMSD-4.2RS) x4
- Induktive sensorerer (SN04-N) x8
- Ledningskanaler x2

Ved analyse av komponentene ble det utformet en antagelse på hvordan systemet var tiltenkt. Med utgangspunkt i antagelsene ble det skissert en grovskisse på hvordan roboten skulle monteres. Figur 3.2 ble benyttet som en provisorisk arbeidsmal gjennom konstruksjonsfasen.



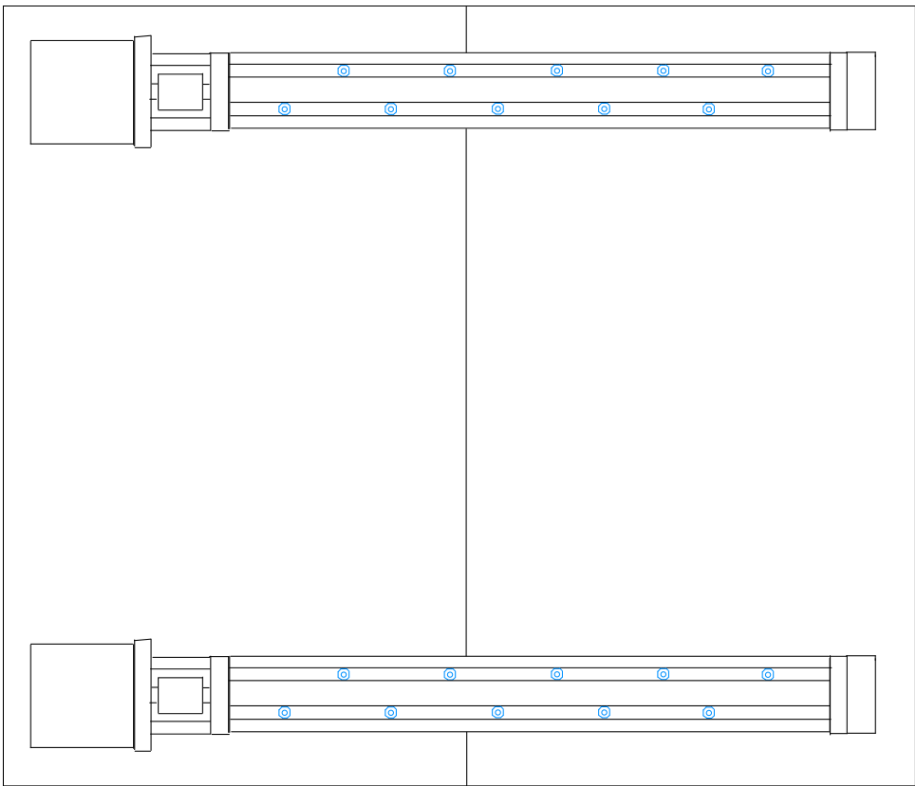
Figur 3.2: Grovskisse av systemet

For å sikre at systemet er stabilt, må den nedre aksen monteres på fast underlag. Det var også ønskelig fra ABB sin side at roboten skulle være mobil, da det ikke var avklart hvor roboten skulle stå i fremtiden. Med det som utgangspunkt ble det kjøpt inn en halvhard-fiberplate også kjent som MDF, med lengde og bredde på henholdsvis 2.44 m og 0.5 m. Platen ble så saget i to på midten for å lage et arbeidsområde på litt over en kvadratmeter, som vist i figur 3.3.



Figur 3.3: Preparering av arbeidsområde

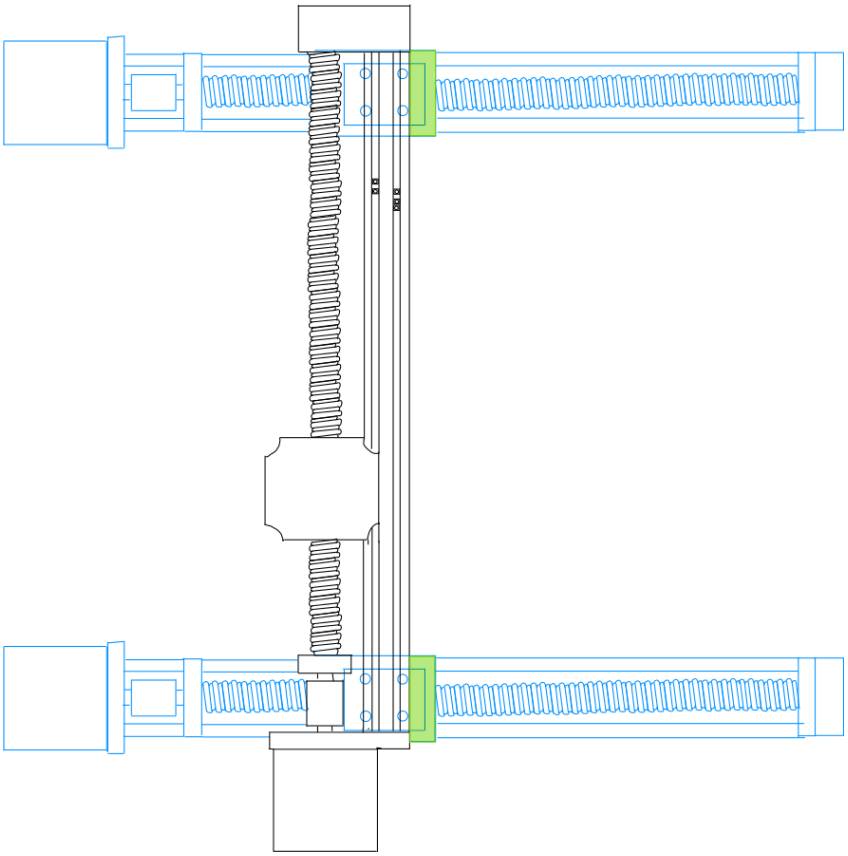
MDF-platene blir festet sammen ved å skru fast akseskinnene direkte i bordet, se figur 2.3 og 3.4. Dette gjøres ved å dra nytte av muttersporene på undersiden av akseskinnene. Det blir gjort ved å bore til sammen 20 hull i platen, ti hull langs hver kant hvor aksene skal gå.



Figur 3.4: Montering av x-akse og plassering av skruene i muttersporet til akseskinnene

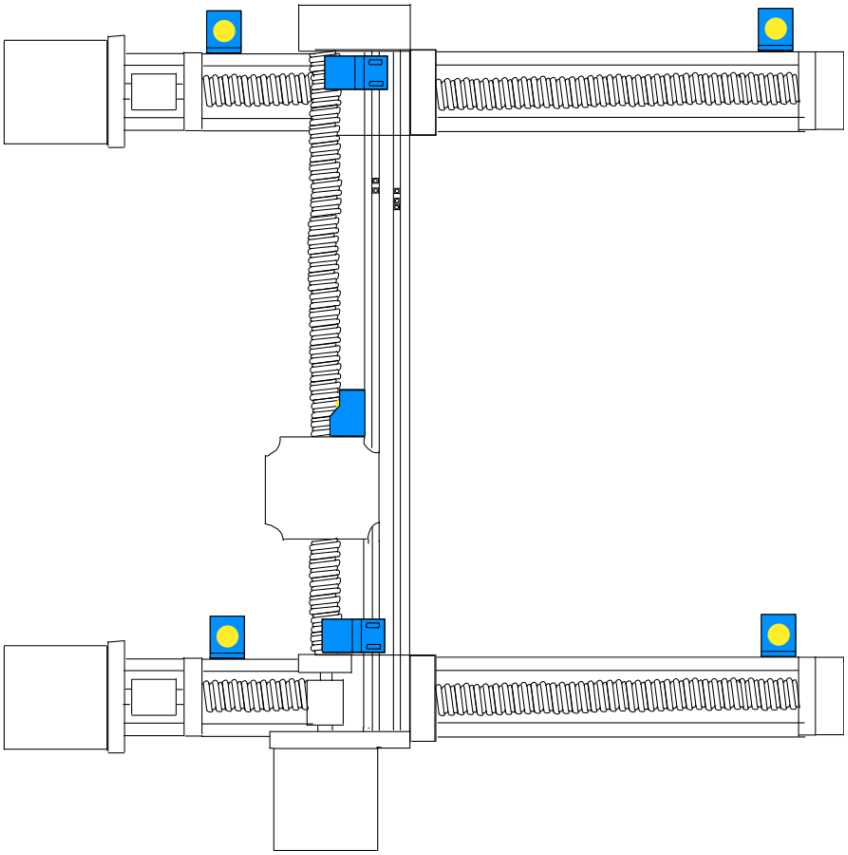
Akseskinnene blir festet ved hjelp av maskinskruer og muttere. For at brettet skulle kunne ligge på flate underlag må maskinskruene felles inn, slik at skruene som fester aksene på undersiden av bordet ikke stikker ut. Ved boring av hull var det ekstremt viktig at det ble utført på en slik måte at de to aksene som går i samme retning blir parallelle, heretter kalt x-akse. Det er for å unngå at roboten kiler seg og går i spenn. Grunnet dette er det også viktig at det blir brukt to stegmotorer for bevegelse i x-aksen. Ideelt sett burde monteringen av aksene bli utført med lasermåler, men ettersom det ikke var tilgjengelig, måtte en meterstokk benyttes og målingene og avlesningene måtte gjøres svært nøye.

Før aksen som skal gå perpendikulært med x-aksen kan bli montert, heretter kalt y-akse, blir aksen som skal gå vertikalt festet på glidebraketten til y-aksen, da dette er enklere å utføre grunnet lettere tilgang til skruene. Etter montering av den vertikale aksen, heretter kalt z-akse, skrus y-aksen fast i glidebraketten med en 90 graders monteringsbrakett som anvist med grønn farge i figur 3.5.



Figur 3.5: Montering av y- og z-akse

Når aksene er korrekt montert, blir det montert induktive sensorer som et sikkerhetstiltak på enden av stagene. De vil fungere som endebrytere for å forhindre at roboten skal kunne kollidere i enden av stagene. Figur 3.6 viser plasseringen til sensorene som blir benyttet for dette systemet.

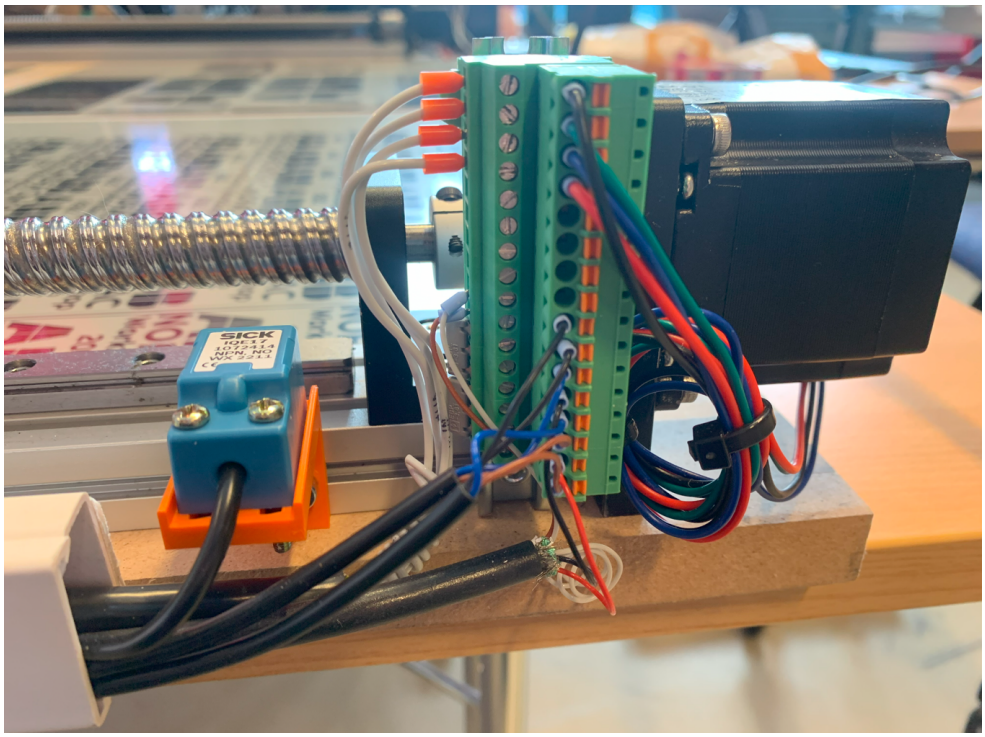


Figur 3.6: Montering av endebrytere

3.1.3 Kabling og elektrisk tilkobling for en kartesisk robot

Etter at alle komponentene som utgjør roboten er skrudd sammen, kan kabler bli lagt mellom de forskjellige komponentene. Det ble bestemt at styrepanelet med all elektronikk skulle monteres direkte på bordplaten på grunn av et ønske om høyere mobilitet fra ABB. Komponentene som benyttes har forskjellig krav til kraftforsyning. De induktive sensoren har en nominell spenning mellom 12-24 volt [C.1], og motorkontrollerene SMSD-4.2RS operer mellom 12-48 volt [7]. Fra manualen til den eksterne I/O-enheten med digitale og analoge inn- og utganger kan en lese at Moxa E1210 har en driftsspenning mellom 12-36 volt [17]. For å ha en kraftforsyning som oppfyller kriteriene til alle komponentene blir det valgt en AC/DC omformer som forsyner 24 volt til systemet ettersom den øvrige grensen til releene og sensorene er på 24 volt. Et 10 amper overbelastningsvern kobles mellom nettspenning og AC/DC omformeren, dette blir også brukt som en av/på bryter for systemet.

Motorene og sensorene som benyttes kom ikke utstyrt med lange nok ledninger til å føres direkte til styrepanelet. Det ble derfor brukt phoenix kontakter til å skjøte kablene som vist i figur 3.7



Figur 3.7: Motor -og sensorkobling

Ettersom at disse kablene skal trekkes rundt brettet blir det benyttet to forskjellige typer kanaler for å holde arbeidsområdet ryddig. Figur 3.8 viser de to kanalene montert.

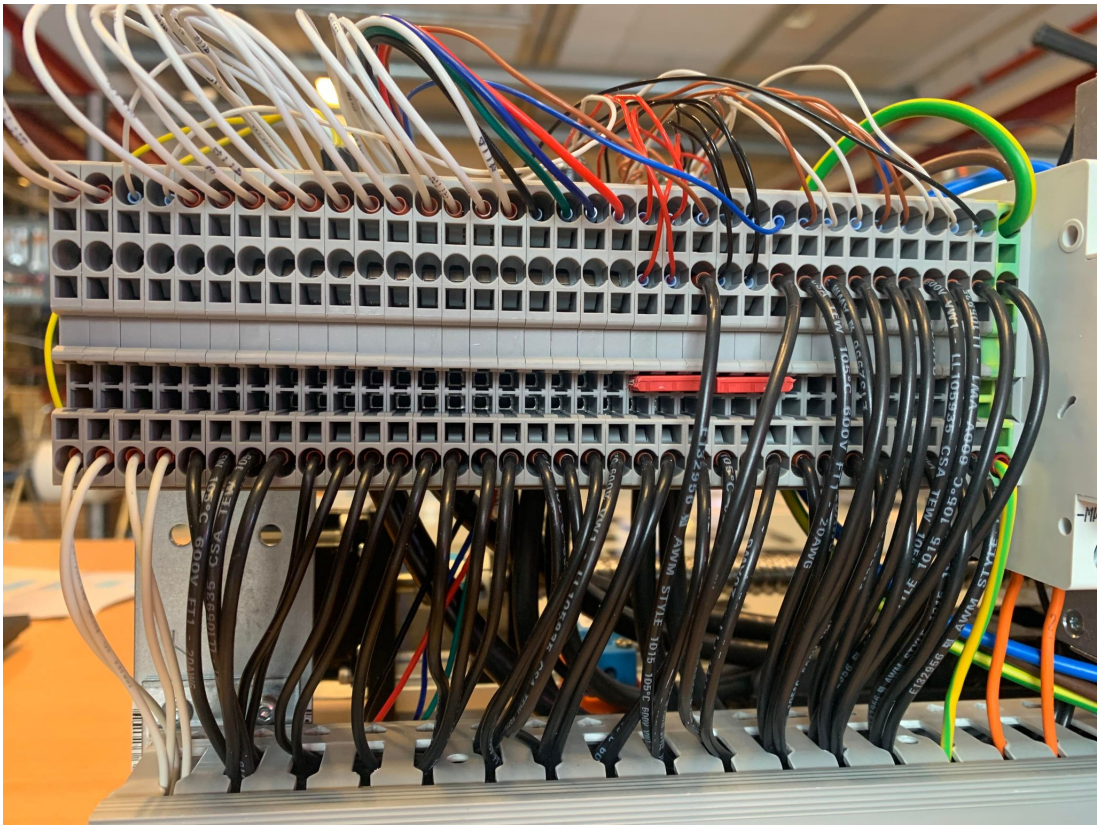
(a) Fast montert kanal for kabel føringer

(b) Bevegelig kanal som flytter seg med systemet

Figur 3.8: Forskjellige kanaler brukt i systemet

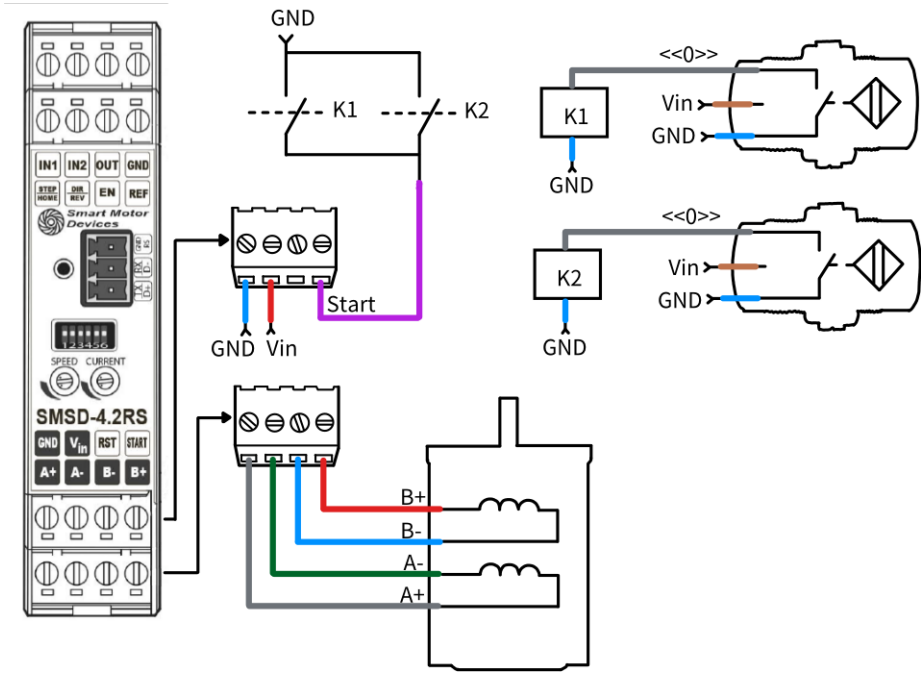
Kanal 3.8a viser en vanlig kabelkanal, men kanal 3.8b er et kablekjede som består av flere ledd og er derfor bevegelig og kan bøyes når roboten beveger seg. Dette er nødvendig for å sørge for at ledningene til de respektive enhetene er lange nok uavhengig av hvor roboten beveger seg. Kanalene fører all kabling til et styrepanelet, men da det ikke var avklart hvor og hvordan styrepanelet skulle konstrueres tidlig i konstruksjonsfasen, ble alle ledningene

terminert i en serie med rekkeklemmer for enklere viderekobling, se figur 3.9.



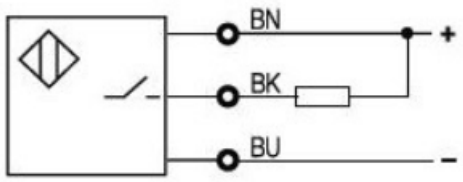
Figur 3.9

Hver stepperkontroller er koblet som vist i figur 3.10



Figur 3.10: Tilkobling av SMSD4.2-RS til en steppermotor og to induktive sensorer inn på *Start* via kontaktorene K1 og K2. Illustrasjonen av kontrolleren er hentet fra [7]

Den induktive sensoren brukes til å oppdage glidebraketten uten direkte kontakt. Når sensoren registrerer glidebraketten, sendes det et signal til kontrollsystemet. Figur 3.11 illustrerer dette ved bruk av en bryter som lukkes når metallobjektet er nærme.



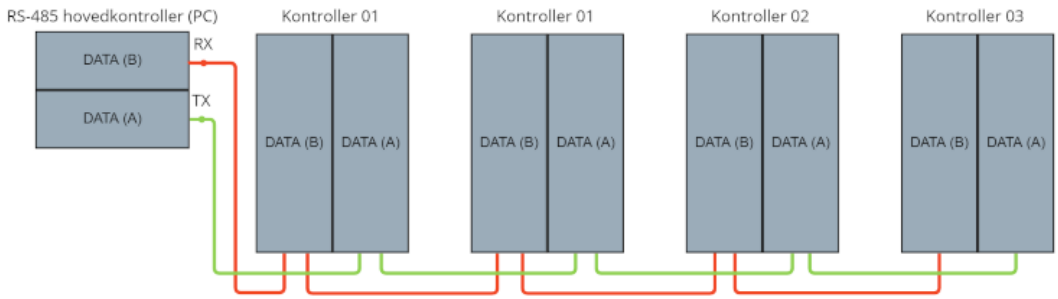
Figur 3.11: Fra C.1

Som vist i figur 3.10 kobles denne signalkabelen via et rele og inn på *Start* inngangen til stepperkontrolleren. Kontrolleren har en innebygget funksjon som stanser en operasjon under utførelse hvis den oppdager en *rising edge* på inngangen til *Start*. I tillegg til at operasjonen blir stanset med et analogt signal er det ønskelig at programmet også får beskjed om at roboten har gått utenfor arbeidsområdet og stoppet. Dette er grunnen til at Moxa IoLogik E1210 I/O-enhet blir brukt.

På grunn av interne koblinger i motordriveren er det et spenningspotensial på 17 V mellom *Start* inngangen på motorkontrolleren og jord. Ettersom signalkabelen fra sensorene ble parallellkoblet med *start* på motordriveren og moxaen, medførte dette at det alltid var en spenning på 17 V på den digitale inngangen til moxaen. Dette førte til at moxaen alltid leser av en høy verdi, uavhengig om endesensoren er høy eller lav, som resulterer i at programmet tror at systemet har kjørt for langt til enhver tid. For å løse dette, må spenningssignalet fra sensoren splittes, slik at moxaen ikke blir parallellkoblet med motordriveren. Dette blir gjort ved å bruke til sammen åtte releer, hvor signal fra hver endesensor blir koblet på A1 til hvert sitt respektive rele. På hvert rele brukes det to *normally open* utganger, hvor den ene *normally open* utgangen kobles til *start* inngangen på motordriveren, og den andre *normally open* utgangen kobles til en digital innputt på moxaen. Ved å anvende denne metoden, hindres spenningen på motordriveren å påvirke moxaen.

3.1.4 Tilkobling av datakabler til maskinvare

Systemet bruker fire stegmotorer og hver stegmotor styres av hver sin motorkontroller. For å kommunisere med kontrolleren ble de koblet opp med RS-485 grensesnitt som vist i figur 3.12. Dette gjør det mulig å kommunisere med alle driverene på en gang slik at hver enkelt driver kan motta forskjellige instrukser, basert på hva slags bevegelse roboten skal utføre. Dette kan gjøres fordi de fire motorkontrollerne har mulighet til å bli adressert med egne adresser slik at de kun tar imot instrukser med sin adresse [7]. To av kontrollerene har fått lik adresse fordi begge styrer samme akse og det er derfor ønskelig at de skal motta og utføre kommandoer til samme tid.



Figur 3.12: RS-485 tilkobling

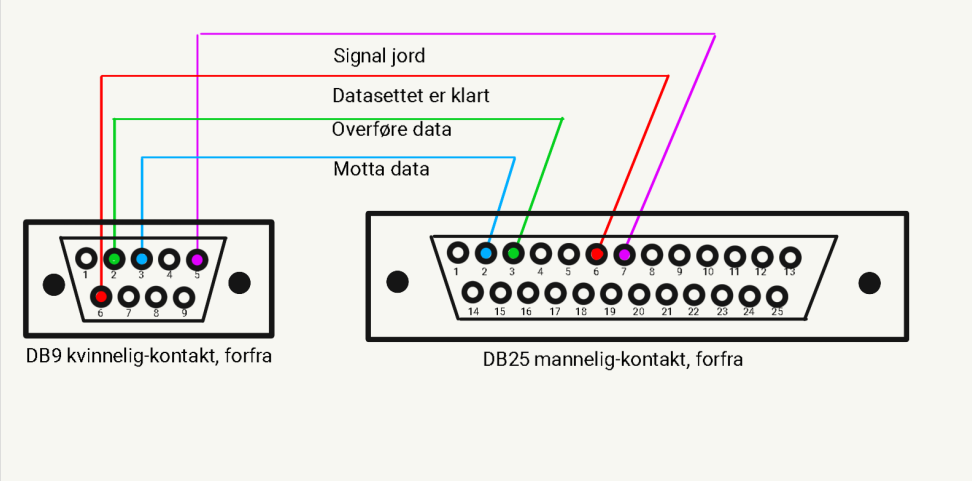
Instrukser blir sendt fra pc-en til kontrolleren som en pakke. Pakken består av en diskret sekvens av ASCII symboler, også kalt en streng. Formatet på pakken er vist i tabell 3.1

Start av pakke	ID adresse	Kommando	Parameter	Slutt av pakke
1	2	3	4	5
:	XX(hex)	CC	NN(desimal)	\r\n

Tabell 3.1: Rammeformat for dataoverføring

Som et eksempel, ASCII strengen ”02MV3200\r\n” er en kommando som blir sendt til kontrolleren med ID-adressen 0x02, hvor MV er instruksjon for kontinuerlig bevegelse og 3200 er parametere som sier hvor mange steg motoren skal bevege seg. På denne måten kan hver kontroller bli styrt individuelt. I tillegg kan ID-adressen 0x00 benyttes for å kommunisere med alle kontrollerne på en gang. Alle kommandoer og parametre som kan benyttes i et brukerstyrt program er hentet fra tabell 6 og 7 i [7].

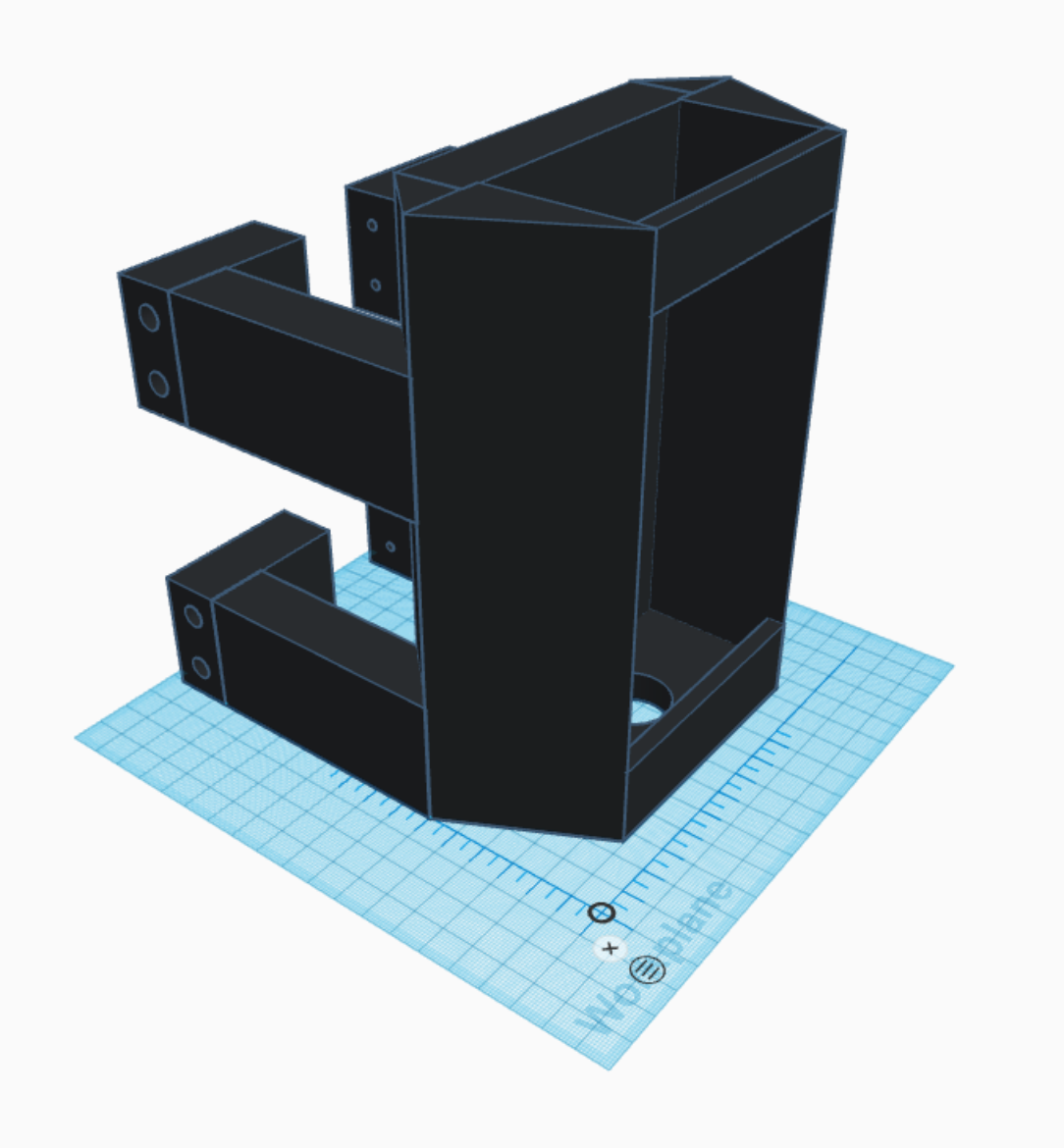
Når datakommunikasjonen mellom kontrollerene og pc-en er ferdig implementert, er det i teorien nå mulig å kjøre roboten ved å gi den instrukser. Før dette blir testet vil den siste datakabelen som kobler Elcometeret til pc-en trekkes. Det er ikke nødvendig med en strømkabel til Elcometeret da denne får strømforsyningen sin av tre 1.5 Volt AA-batterier. Elcometeret benytter seg av en RS-232 kommunikasjonsmodul via en DB25 kontakt. Pinnekonfigureringen er vist i figur 3.13.



Figur 3.13: RS-232 tilkobling

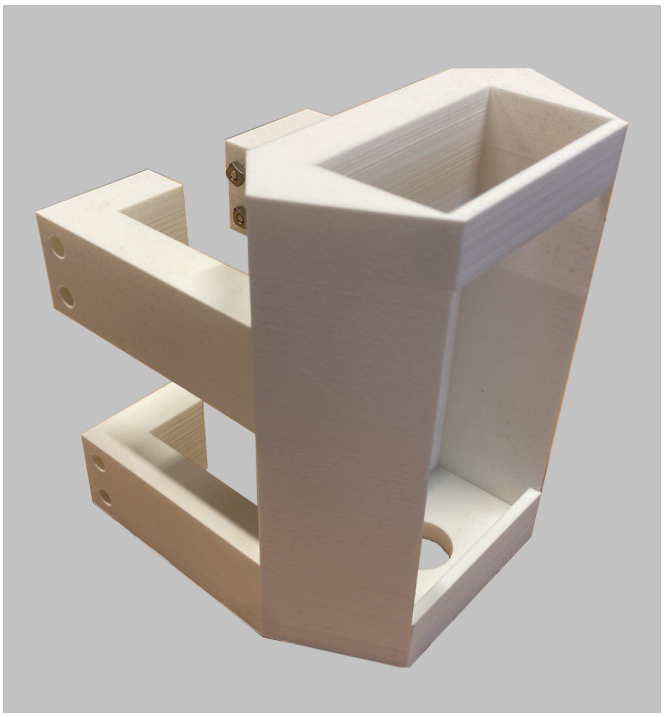
3.1.5 3D-printing av nødvendige monteringsløsninger

Elcometer 355 modulen er designet for håndholdt bruk. Det gjør at modulen kommer uten en monteringsbrakett som kan brukes til systemet. For å løse denne utfordringen ble det tatt i bruk 3D-printing via Tinkercad. Braketten er laget på en slik måte at den kan holde Elcometeret stabilt mens den beveger seg over brettet. Den er også designet med et hull i bunn av braketten for enkel inngang og utgang hvis det skulle være behov. Figur 3.14 viser 3D-modellen i Tinkercad før produksjon.



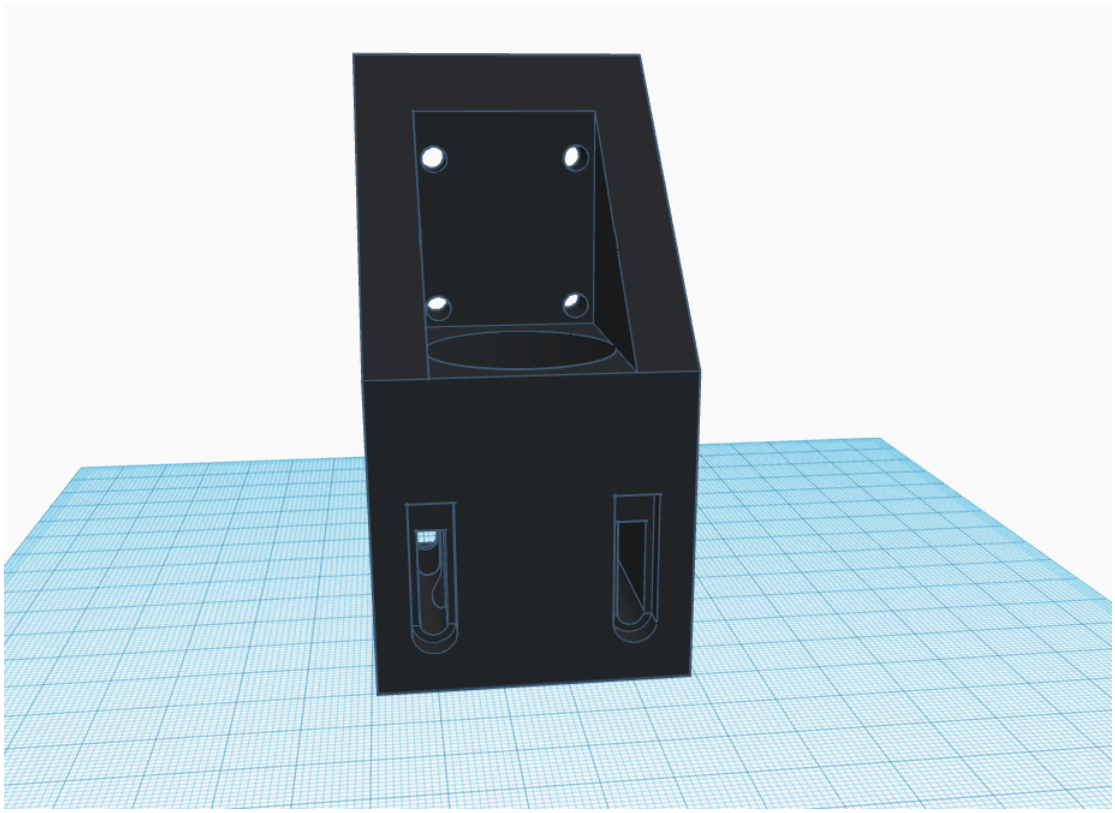
Figur 3.14: 3D modell av brakett til Elcometer

3D-modellen blir så lagret som en STL-fil og sendt videre til 3D-printeren som lager braketten. Materialet braketten er laget av heter *Tough PLA* som er lett og kraftig nok til å gi ønsket funksjonalitet [21]. Det ble gjort nøye målinger for å forsikre seg om at avstanden til hver enkelt del, og skruehull stemte overens med skruehullene som var å finne på staget til z-aksen.



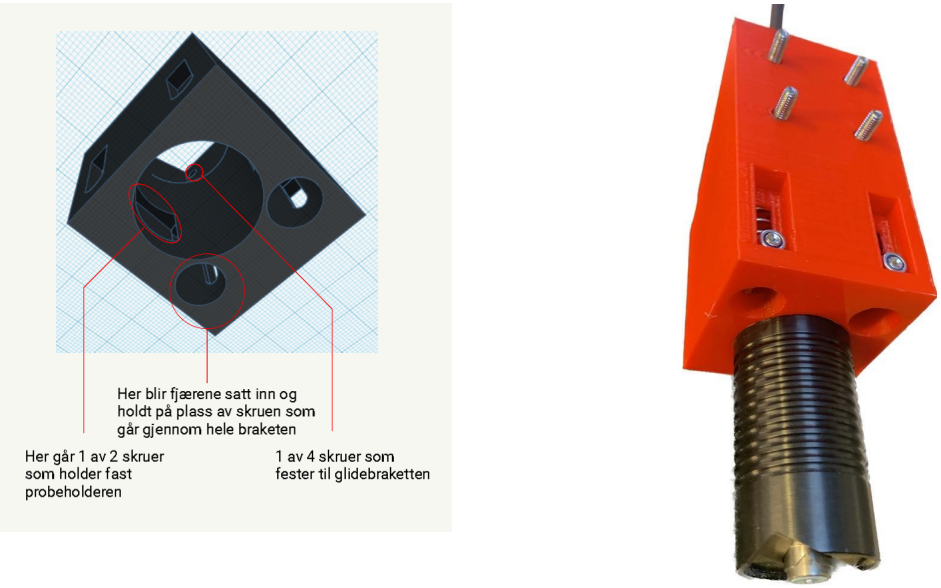
Figur 3.15: Ferdig 3D-printet brakett til Elcometer

I tillegg til selve modulen er også proben laget for å utføre håndholdte målinger. Det ble derfor også laget en brakett som kunne holde probehodet på plass under bevegelse av roboten. 3D-modellen som ble designet i Tinkercad er vist i figur 3.16



Figur 3.16: Modell for sensorholder

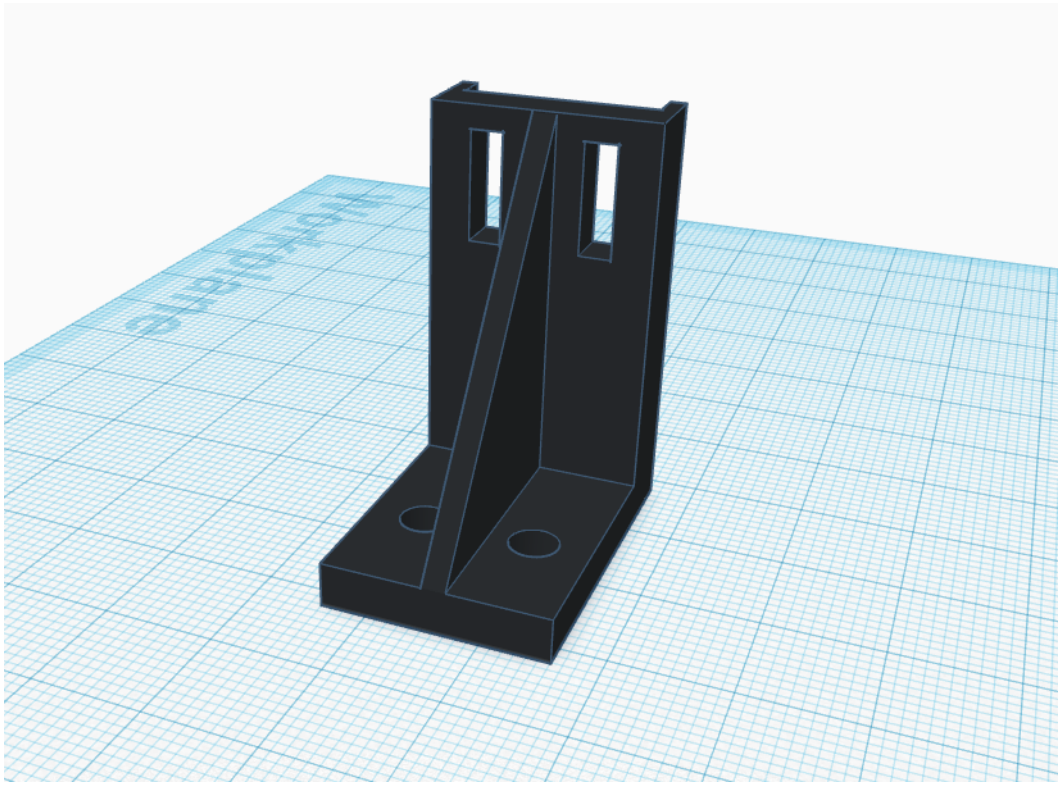
Funksjonen til braketten er å holde probehodet på plass mens det beveger seg vertikalt eller horisontalt i arbeidsområdet. Dette gjøres ved å skru braketten fast i glidebraketten som drives langs z-aksen. Som et ekstra sikkerhetstiltak, for å ta vare på sensoren, ble ekstra demping implementert i braketten. Det er gjort ved hjelp av fjærer som ble plassert i bunnen av braketten, mellom skruene som holder probenhåndtaket til braketten, og selve braketten. Avlange hull blir brukt slik at skruene skal kunne bevege seg vertikalt uten andre hindringer enn fjærene. En metallskive blir plassert mellom fjæren og skruen for å unngå at gjengene på skruen skraper mot fjæren.



(a) Modell for sensorholder sett fra bunnen (b) Sammensatt og klar for montering på z-aksen

Figur 3.17

Etter en del kjøring og testing ble det oppdaget at monterings brakettene for endesensorene var av dårlig kvalitet, og med et design som ikke passet helt til dette systemet. På grunn av lav kvalitet på plastikken, og et flatt underlag førte det til at sensoren kunne ende opp med å stå skjevt. Dette skyldes kontakt med brukeren eller vibrasjoner som oppstod under kjøring. Dette resulterte i at sensoren kom for langt unna glidebrakettene slik at den ikke klarte og plukke opp at roboten var utenfor arbeidsområdet. Dermed ville ikke sensoren sende et stoppsignal til driveren som igjen resulterte i at motoren kolliderte med sluttstykket til staget og motoren fortsetter å rotere. Det ble derfor laget en forbedret modell for endesensorene for å motvirke dette problemet som vist i figur 3.18.



Figur 3.18: Modell for induktiv sensorholder

Funksjonen til den nye modellen er lik den gamle, men det er nå mulig å flytte sensoren lengre fremover eller bakover om man skulle trenge det i tillegg til at det er løftet opp kanter på siden av braketten som hindrer sensoren i å rotere.

3.2 Programvare

I dette delkapittelet blir utvikling av programmet for styring av systemet og behandling av data presentert. Delkapittelet viser de forskjellige stegene systemet skal utføre og hvordan de blir utviklet og implementert.

3.2.1 Innledning

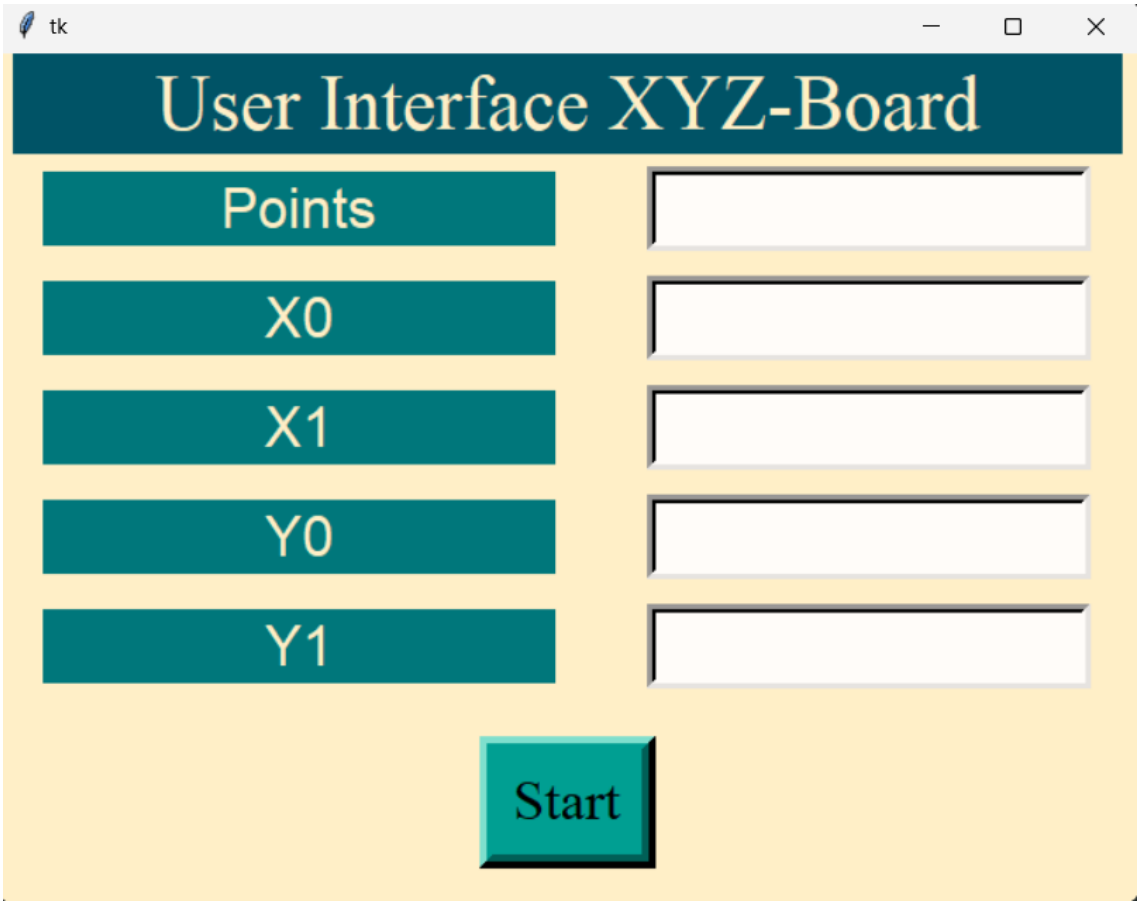
For at roboten skal kjøre delvis etter ønske fra brukeren blir informasjon om måleområdet og oppløsning hentet og gjort om til en rekke med motorstyringskommandoer for å på en nøyaktig og effektiv måte utføre målinger over arbeidsområdet. Det er også viktig å kunne ta vare på data som er samlet inn for videre vurdering og sammenligning. Funksjoner for å oppnå dette blir presentert i dette kapittelet

3.2.2 Innhenting og håndtering av brukerdefinerte verdier

Programmet er konstruert på en slik måte at brukeren sender informasjon en gang før oppstart. Etter det vil systemet kjøre autonomt til operasjonen er utført. Variablene brukeren oppgir er følgende.

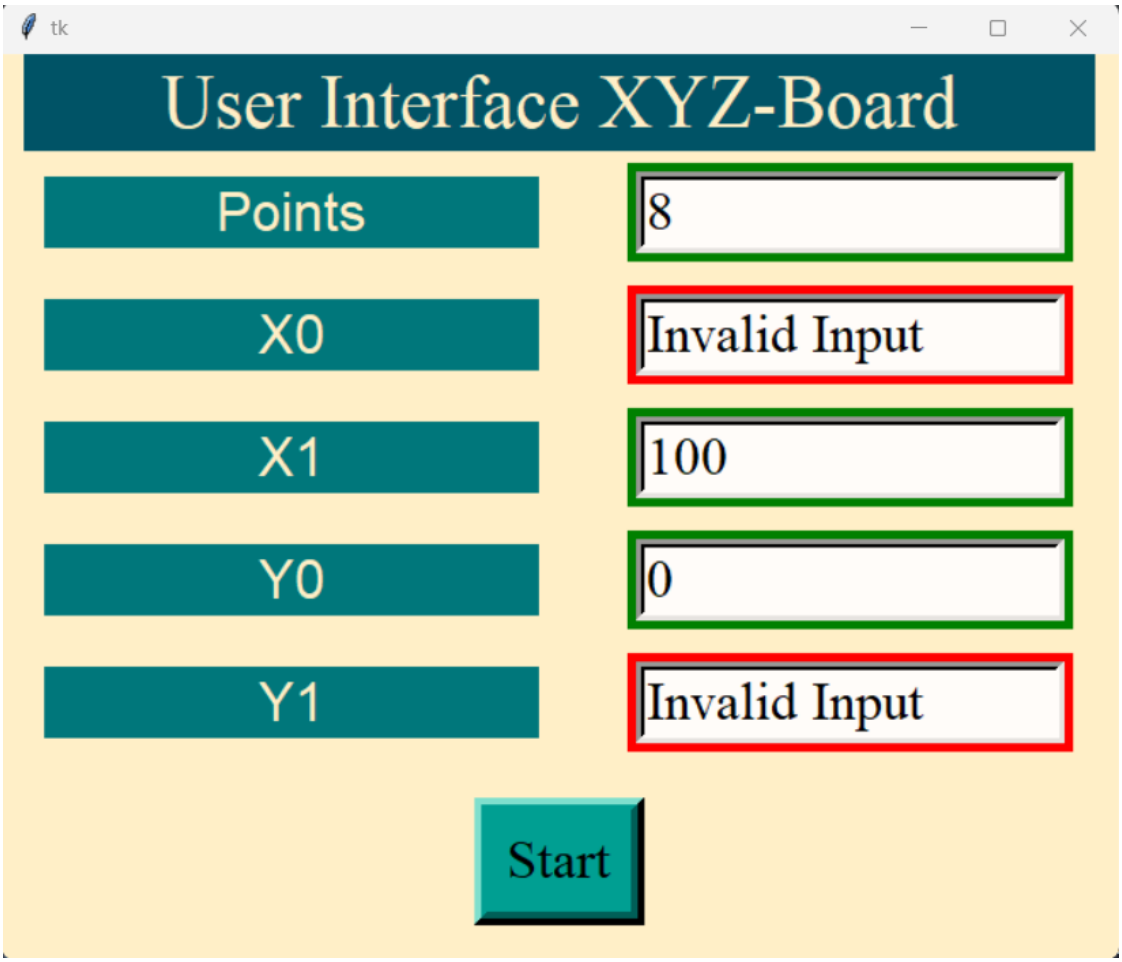
- Antall målinger
- Startpunkt i x-retning, x_0
- Sluttpunkt i x-retning, x_1
- Startpunkt i y-retning, y_0
- Sluttpunkt i y-retning, y_1

De hentes via et grafisk brukergrensesnitt som er vist i figur 3.19



Figur 3.19: Grafisk brukergrensesnitt

Når startknappen fra figur 3.19 trykkes vil hver verdi i tekstboksene bli forsøkt konvertert om til et desimaltall. Hvis denne konverteringen genererer en feil, som kan oppstå hvis en for eksempel har sendt inn en bokstav eller en tom tekstboks, vil verdien bli gjort om til *None*. På denne måten kan en sjekke om det finnes en *None* i lista med inngangsverdier. Hvis det er tilfelle vil ikke programmet starte, GUI-en vil bli oppdatert, men de gyldige verdiene vil bli tatt vare på for enklere utfylling ved neste forsøk.



Figur 3.20: Eksempel hvor noen verdier var gyldige og noen ugyldige ved start av programmet

Når alle verdiene blir konvertert om fra tekst til desimaltall blir de brukt til å beregne overflaten det skal måles over, hvor måleproben skal starte og hvor mange målinger som vil bli gjort på hver akse. I tillegg blir vinduet lukket for å forhindre brukeren fra å starte programmet på nytt under utførelse. Første beregning er lengden i x-retning. Det blir gjort på følgende måte

$$|x| = x_1 - x_0 \tag{3.1}$$

Og samme blir gjort for lengden i y-retning. Når programmet vet hvor stor overflaten er, fordeles målingene ut ifra en prosentvis fordeling av aksenes størrelse. Beregningen blir gjort slik.

$$x[\%] = \frac{|x|}{|x| + |y|} \tag{3.2}$$

For å finne antall målinger som skal utføres i hver akse må det settes noen betingelser. Antall målinger på hver akse kan ikke være et desimal og fordelingen av målingene skal kunne forme enten et rektangel eller et kvadrat. Det vil si at det ikke skal være ukomplette linjer med målinger. For å ha fulle linjer kan en enten legge til målinger for å fylle en linje, eller trekke fra de resterende målingene på en ukomplett linje. Det er ikke ønskelig å redusere antall målinger brukeren ber om for å oppfylle betingelsene, da dette kan svekke kvaliteten på dataen. Av den grunn ble det bestemt at det alltid skal legges til målinger for å fylle en linje hvis ikke antallet målinger som ble gitt av brukeren kan fordeles som et rektangel eller et kvadrat.

Denne funksjonen kan uttrykkes matematisk på følgende måte med et eksempel. Tekstboksene i figur 3.19 blir gitt følgende verdier.

$$\begin{aligned} \text{Antall målinger} &= 8 \\ x_0 &= 0 \\ x_1 &= 100 \\ y_0 &= 0 \\ y_1 &= 100 \end{aligned}$$

Den prosentvise fordelingen av målinger på hver akse blir da.

$$\begin{aligned} percent\_x\_dir &= \frac{(100-0)}{(100-0)+(100-0)} \\ &= 0.5 \\ percent\_y\_dir &= \frac{(100-0)}{(100-0)+(100-0)} \\ &= 0.5 \end{aligned} \tag{3.3}$$

Resultatet blir naturligvis at målingene skal fordeles likt på aksene da arealet gitt av brukeren i dette eksemplet former et kvadrat. Antall nødvendige målepunkter blir beregnet ved å sette opp ligningen under.

$$\begin{aligned} 0.5 \cdot 0.5 \cdot (i)^2 &\geqslant 8 \quad , i = \mathbb{N} \\ 0.25 \cdot (i)^2 &\geqslant 8 \\ i^2 &\geqslant \frac{8}{0.25} \\ i &\geqslant 4\sqrt{2} \\ i &= 6 \end{aligned} \tag{3.4}$$

Som gir følgende løsning for ligning 3.4

$$0.5 \cdot 0.5 \cdot (6)^2 = 9 \tag{3.5}$$

Det blir her lagt til en ekstra måling sammenlignet med hva brukeren ba om for å kunne måle jevnt over et kvadrat. Antall målinger for hver akse blir da.

$$\begin{aligned} x\_points &= i \cdot percent\_x\_dir \\ &= 6 \cdot 0.5 \\ &= 3 \end{aligned} \tag{3.6}$$

Tilsvarende blir resultatet for $y\_points = 3$, som tilsvarer hvor mange målinger som skal gjøres på y-aksen. Ettersom det er ønskelig å spre målingene utover arbeidsområdet gitt

fra brukeren, er det naturlig å flytte første måling litt ut fra origo. Dette gjøres ved å tenke at det er to punkter mellom hver måling, og ett punkt fra første og siste måling langs hver akse. Forskyvningen til første måling i både x- og y-retning blir da.

$$\begin{aligned} mm\_x\_offset &= \frac{x_1 - x_0}{x\_points \cdot 2} \\ &= \frac{100 - 0}{6} \\ &= 16.67mm \end{aligned} \tag{3.7}$$

Etter å ha testet kjøring på stagene ble det målt at 36 grader rotasjon på stegmotoren tilsvarer 1 mm med forflytning av sensorhodet. Gradene motoren skal rotere til første målepunkt blir da.

$$\begin{aligned} deg\_x\_offset &= 16.67 \cdot 36 \\ &= 600.12^\circ \end{aligned} \tag{3.8}$$

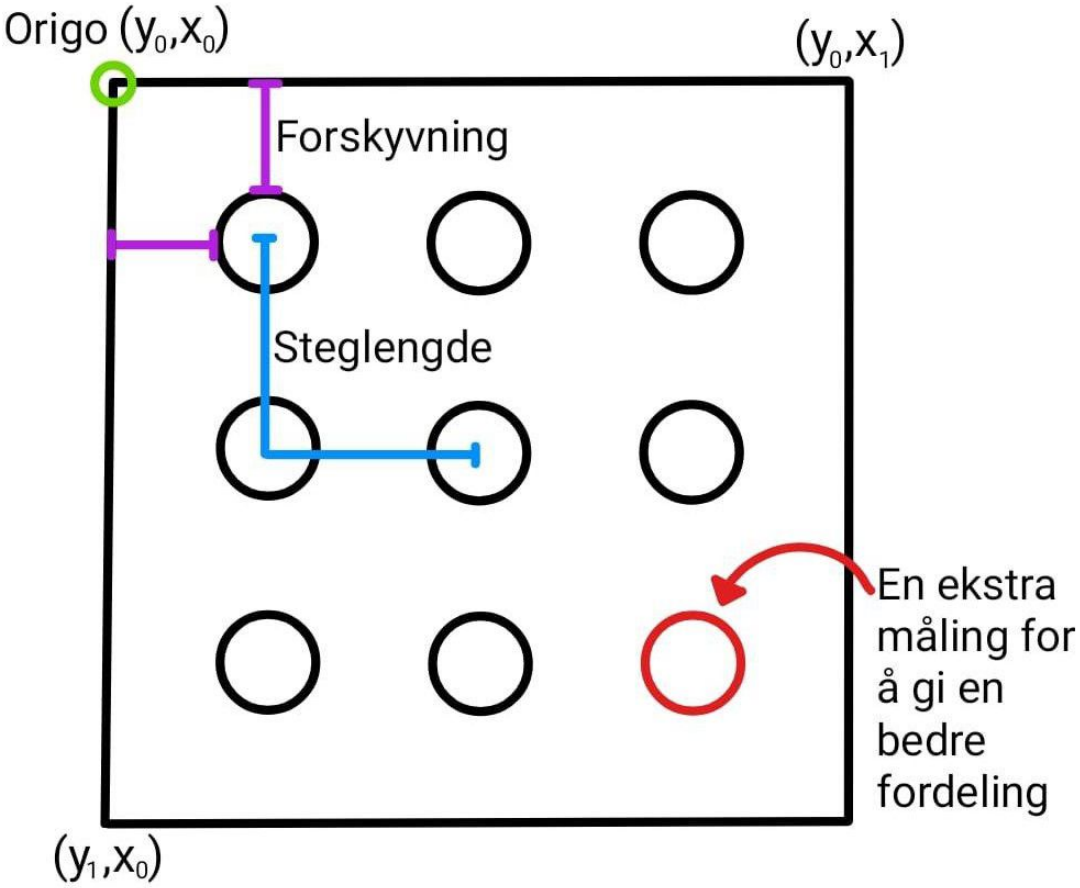
Ettersom spredningen av målingene er tenkt å være to punkter mellom hver måling, blir steglengden mellom punktene.

$$\begin{aligned} mm\_x\_move &= mm\_x\_offset \cdot 2 \\ &= 16.67 \cdot 2 \\ &= 33.34mm \end{aligned} \tag{3.9}$$

Den siste beregningen som skal gjøres er gradene motoren skal forflyttes mellom hvert målepunkt. Roteringsgradene i x- og y-retning tilsvarer da.

$$\begin{aligned} deg\_x\_move &= 33.34 \cdot 36 \\ &= 1200.24^\circ \end{aligned} \tag{3.10}$$

Resultatet av disse beregningene gir en fordeling av målingene som vil se noe ut som vist i figur 3.21



Figur 3.21: Illustrering av hvordan målingene blir fordelt jevnt utover en plate.

3.2.3 Utvikling av timere for presis dataoverføring

SMSD-4.2RS har mulighet for tilbakekobling i form av en beskjed som kan returneres hvis brukeren ber om det, se tabell åtte, side ni [7]. Dette kan utnyttes på en slik måte at en aktiverer dataoverføring etter å ha sendt én kringkastningsintruksjon, *EM*. Kontrolleren vil da sende meldingen *E14* til brukeren etter at en operasjon er utført. Når dette signalet blir mottatt vet man at en instruksjon er fullført. Utfordringen med kontrollerene er at de kan sende meldingen fordelt på to datapakker. Dette gjør at programmet ikke klarer å dekode meldingen på grunn av manglende start- og sluttbit. For at programmet skal kjøre mer stabilt blir derfor timere brukt til å regulere om en operasjon er utført.

Fra dokumentasjonen til SMSD-4.2RS blir det oppgitt at driveren har grenseverdier for hastigheten mellom 1 og 10000 pulser per sekund (pps). Stegmotoren har 200 steg per rotasjon og kontrolleren har en mikro stegfaktor (MSF) på $\frac{1}{1}$, $\frac{1}{2}$, $\frac{1}{4}$ og $\frac{1}{16}$ [7]. Antall steg per grad blir derfor beregnet slik.

$$\text{Steg per grad} = \frac{200 \cdot MSF}{360^\circ} \tag{3.11}$$

For å bruke dette til å beregne tiden det tar for sensoren å forflytte seg, settes følgende ligning opp

$$\text{Tid for utførelse}[sek] = \frac{\text{Vinkel[grader]}}{\text{Hastighet[pps]}} \cdot \text{Steg per grad} \tag{3.12}$$

Denne verdien vil bli brukt på en slik måte at når en kontroller har fått beskjed om å rotere en bestemt vinkel med en gitt hastighet, vil programmet vente med å sende neste instruks til tiden er ute. Ved tilnærmet optimal kjøring vil et slikt estimat være godt nok til å gi god programflyt og kjøring, men hvis motoren av en eller annen grunn bruker lengre tid på å rotere den bestemte lengden, vil ikke programmet kunne vente på en slik forsinkelse. Hvis kontrolleren er ferdig med oppgaven før den estimerte tiden, vil heller ikke programmet gå videre umiddelbart, men vente til tiden går ut. Det kan derfor også være lite tidseffektivt å benytte seg av en slik timer.

3.2.4 Utvikling av automatisk tilkobling til I/O-porter

Både stepperkontrollerene og Elcometeret er koblet opp via USB-innganger på en PC. USB-inngangene gir en COM port (communication port), som er et I/O grensesnitt. COM-porter blir ofte referert som en serieport. For at programmet skal sende beskjeder til riktig USB-port blir funksjonen *comport_provider* brukt. Den finner ut hvilke COM-port som er koblet til hvilken enhet. Det gjøres som vist under.

```
133     def comport_provider(self):
134         ports = list(serial.tools.list_ports.comports())
135         for port in ports:
136             if 'Prolific USB-to-Serial Comm Port' in port.description:
137                 self.elcometer_comport = port.device
138             elif 'USB-SERIAL CH340' in port.description:
139                 self.stepper_comport = port.device
```

Kode 3.1: StepperCmd : Metode for å finne korrekt kommunikasjonsport

Programmet benytter seg av en innebygget modul i Python som heter *serial*. Denne modulen er et verktøy som gjør det mulig å kommunisere med ekstern seriell maskinvare. COM-porten som tildeles USB-enheten tilkoblet PC-en varierer fra maskin til maskin. Det er derfor nødvendig å finne ut hvilke COM-port som USB-enhetene har fått etter at de ble koblet til. Funksjonen som søker etter riktig COM-port, starter med å lage en liste over alle COM-portene tilgjengelig på PC-en. Deretter går den gjennom beskrivelsen til hver port for å finne riktig navn til USB-enheten. Når programmet har funnet riktig COM-port, vil den bruke denne porten når den skal kommunisere.

Det er viktig å merke seg at portbeskrivelsen er spesifikk for den tilkoblede kommunikasjonskabelen og det er derfor nødvendig å bruke samme komponent som funksjonen er laget for. Hvis ikke vil en kunne få problemer med kommunikasjonen. Som nevnt i kapittel 2.2 er det noen parametere som må spesifiseres for å sikre at kommunikasjonen blir riktig. Elcometerets COM-port blir konfigurert på følgende måte.

```
5     self.ser.baudrate = 9600
6     self.ser.port = serialport_name
7     self.ser.stopbits = serial.STOPBITS_ONE
8     self.ser.parity = serial.PARITY_NONE
9     self.ser.bytesize = serial.EIGHTBITS
```

COM-porten som stepperkontrollerene er koblet til blir konfigurert slik.

```
5       self.ser.baudrate = 9600
6       self.ser.port = serialport_name
7       self.ser.stopbits = serial.STOPBITS_ONE
8       self.ser.parity = serial.PARITY_EVEN
9       self.ser.bytesize = serial.EIGHTBITS
```

Kommunikasjonen mellom moxa I/O-enheten og python-programmet skjer ved hjelp av python-klassen GetSensorData. Dette programmet henter informasjon fra en IPv4 adresse, som blir konfigurert ved hjelp av et ethernet adapter. Adressen blir konfigurert til å være 192.168.127.254 der 254 er moxa-enheten sin adresse. Informasjonen blir så hentet av python, dette blir kalt et API. Informasjonen blir så konvertert til JSON (JavaScript Object Notation) og lagret i klassen.

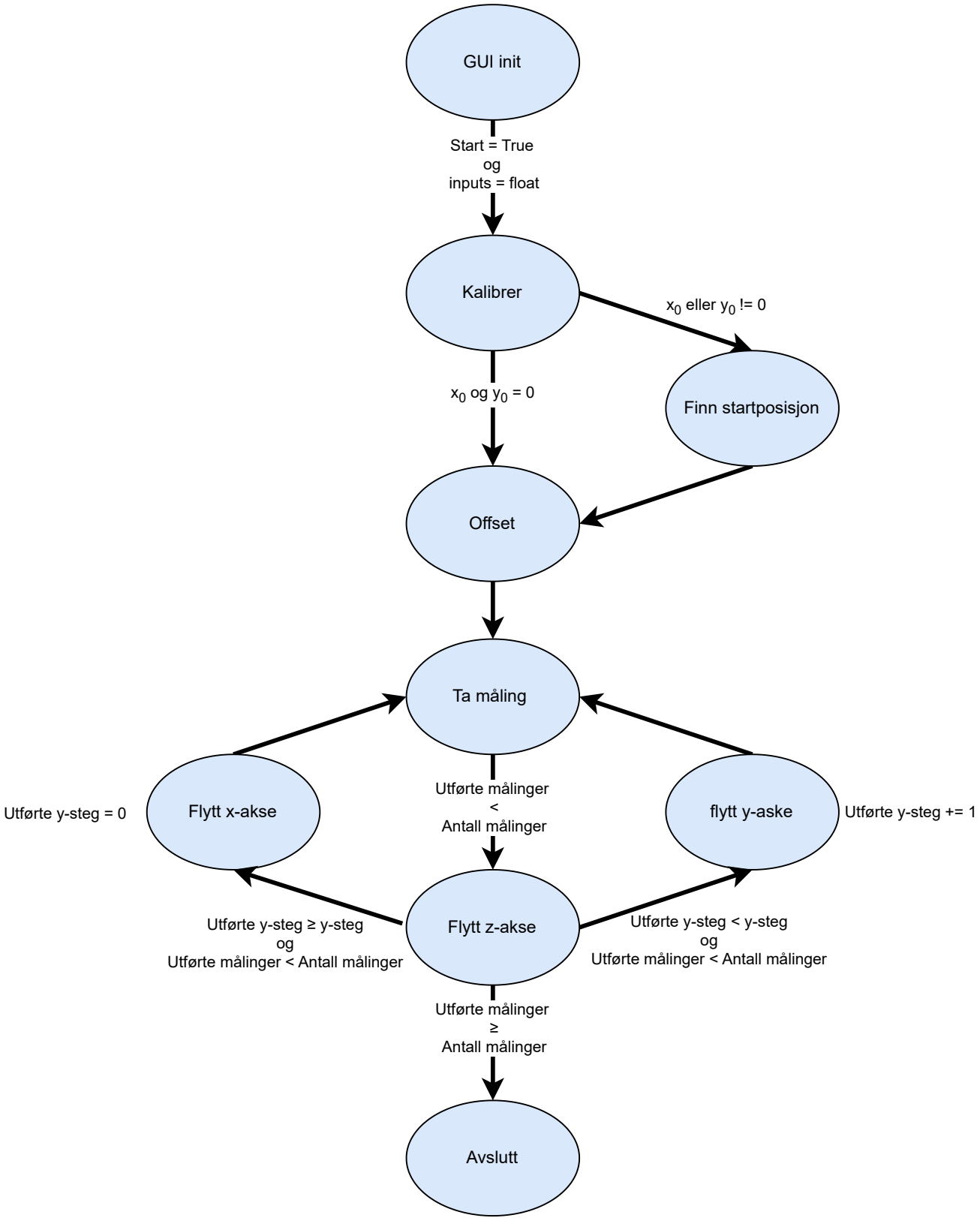
I kode utdraget 3.2 er det *self.get_data* som henter informasjonen fra den riktige plassen. Her vil det si at */API* indikerer at API-et blir brukt. */slot/0* refererer til den spesifikke modulen inne i API-et. */io* betyr at det er en *input/output* og */di* spesifiserer at det er en digital input. Så blir dette konvertert til JSON og informasjonen for *io* og *di* hentet og lagret i variabelen *self.DI*.

```
35          def get_DI(self):
36          self.get_data(api_address_extension="/api/slot/0/io/di")
37          if not self.connection_error:
38              self.convert_received_data()
39              self.DI = self.json_data['io']['di']
```

Kode 3.2: GetSensorData : Metoden *get_DI* henter ut informasjon fra moxaen

3.2.5 Utvikling av kommandosekvenser for kjøring

For at roboten skal utføre ønsket bevegelse må instruksene til kontrollerene sendes i riktig rekkefølge og til riktig tid. Hvis denne timingen blir feil, kan det få uventede og uønskede konsekvenser. Det er derfor nødvendig å bygge opp programmet slik at hver sekvens som sender motorinstrukser blir helt ferdig før neste sekvens blir startet. Tilstandsdiagrammet under viser i hvilken rekkefølge roboten skal utføre en operasjon.



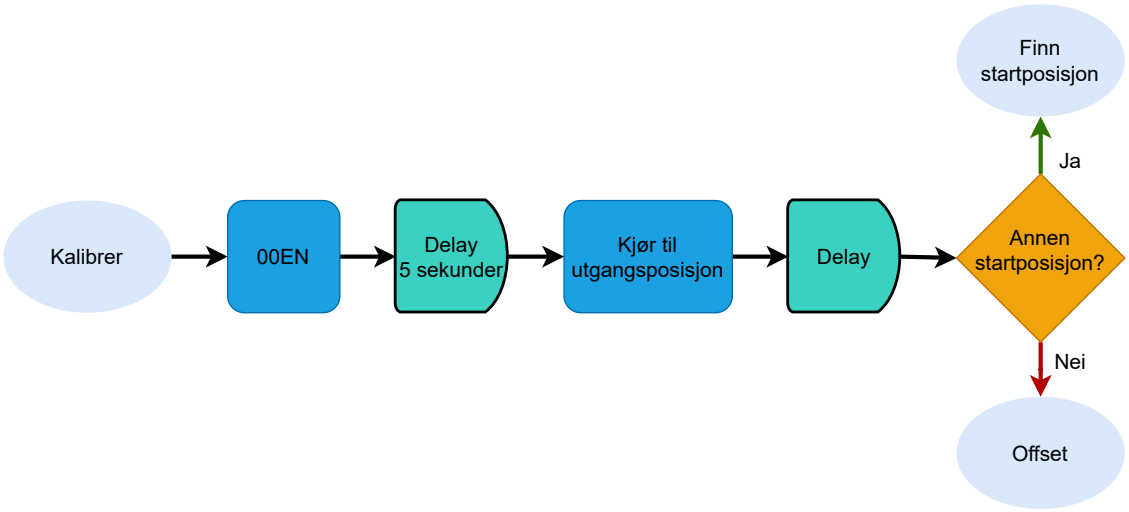
Figur 3.22

Hver tilstand består av mange motorkommandoer. De blir bygget opp og sendt til kontrolleren som forklart i kapittel 3.1.4. Siden enkelte av de eksterne enhetene skal leses av kontinuerlig er det problematisk at programmet utføres sekvensielt og stopper når den

venter på en motorbevegelse. For å oppnå kontinuerlig avlesning av eksterne enheter må forskjellige python-funksjoner kjøre samtidig. Det kan gjøres ved bruk av *multithreading*. En tråd, eller en *thread*, kan betraktes som en sekvens av instruksjoner som utføres av CPU-en og flere tråder kan kjøre parallelt på en eller flere CPU-er [23, s. 557]. Dette gjør at applikasjonen kan utføre oppgaver på separate tråder og opprettholde en akseptabel responstid i tillegg til å unngå at det blir blokkert av en enkel tråd der oppgaven tar lang tid å utføre.

De to funksjonene det er nødvendig å kjøre i paralell er avlesning av Elcometeret og endbryterne. Grunnen til det, er at endebryterene er en del av sikkerhetssystemet som forhindrer roboten i å kjøre utenfor arbeidsområdet. For at den funksjonen skal ha noe effekt må den kunne aktiveres uansett hvilke tilstand roboten befinner seg i. Kontinuerlig avlesing av Elcometeret vil si at man lytter til COM-porten som er tilkoblet og ser når det dukker opp en datapakke som er lengre enn 24 symboler. Denne datapakken er en UTF8-kodet streng som så blir dekodet. Den dekodede strengen vil inneholde målenummer, måletykkelse og enheten til måletykkelsen.

Kalibrering er det første som skal skje etter at brukeren har gitt gyldige verdier og trykket start. Den starter med å sende en instruks til alle kontrollerene om å strømsette stegmotorene. Det gjøres med *00EN*. Deretter venter den i fem sekunder. Denne forsinkelsen er for at de to funksjonene som skal kjøre parallelt rekker å initialisere før motorene begynner å rotere.



Figur 3.23: Flytskjema viser sekvensen motorkommandoene blir sent til kontrollene for å kalibrer systemet

Grunnen til at roboten kalibreres er fordi det ikke er mulig å kjenne til systemets faktiske posisjon i sanntid. For å komme rundt det problemet blir roboten forflyttet til et hjørne av brettet før den begynner å ta målinger. Dette gjør det mer forutsigbart og enklere å utvikle en algoritme for å forflytte roboten på samme måte, med samme mønster hver gang. Pådragene til de lengste aksene er den lengste mulige forflytningen roboten kan gjøre. Siden en rotasjon tilsvarer 10 mm forflytning, og det er mulig å forflytte roboten 985 mm. Det vil da si at maksimal rotasjon blir

$$985mm \cdot 36^\circ = 35460^\circ \tag{3.13}$$

Etter at alle pådragene er satt, starter en nedtelling på omtrent den tiden det tar å bevege

roboten fra den ene siden av brettet, til den andre. Dette er fordi det er den lengste tiden programmet kan bruke på å komme til kalibrert posisjon. For å gå fra den ene siden av brettet til den andre, må motoren som nevnt rotere 35460 grader i x- og y-retning. Fra ligning 3.12 får en.

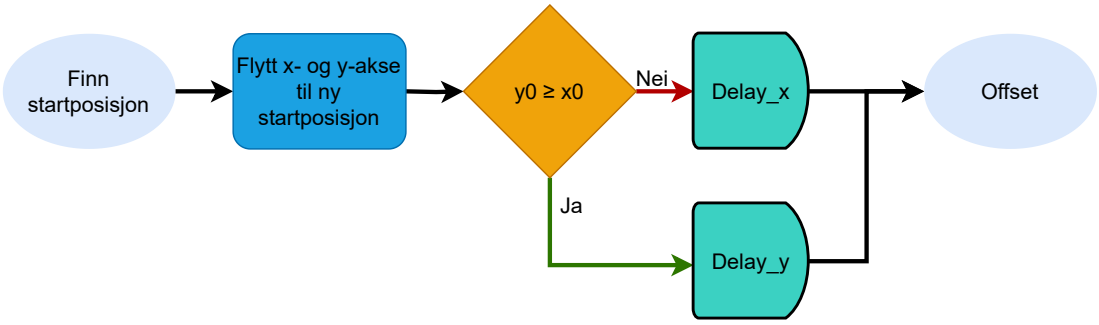
$$delay[sek] = \frac{35460}{pps} \cdot \frac{200 \cdot MSF}{360} \tag{3.14}$$

Dette resulterer i en kjøring som ikke er tidsoptimal, men siden man ikke vet posisjonen til systemet ved oppstart, er dette en løsning som gir et stabilt resultat. Når nedtellingen er ferdig, sjekkes startverdiene for å finne ut om brukeren ønsker å starte målingene i et annet punkt enn (0,0). Ønsker brukeren et annet utgangspunkt enn origo vil systemet flytte måleproben til ønsket startposisjon, hvis ikke vil den umiddelbart forskyve proben slik at målingene får et offset.

Når roboten skal forflyttes til startposisjonen blir startverdiene x_0 og y_0 multiplisert med 36 for å beregne rotasjon for motorene som styrer x- og y-aksen.

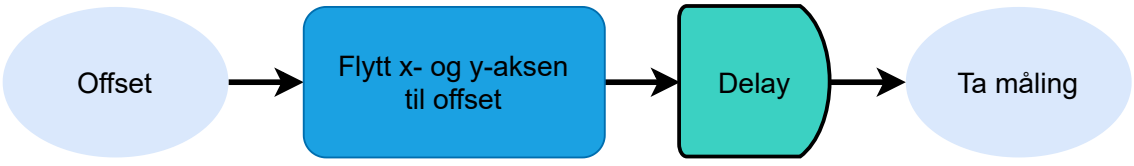
$$\text{deg\_x\_start} = x_0 \cdot 36 \tag{3.15}$$

Beskjeden om å rotere sendes så til motorkontrollerene som illustrert i figur 3.24



Figur 3.24: Roboten flyttes til startposisjon ved å gi et pådrag på aksene

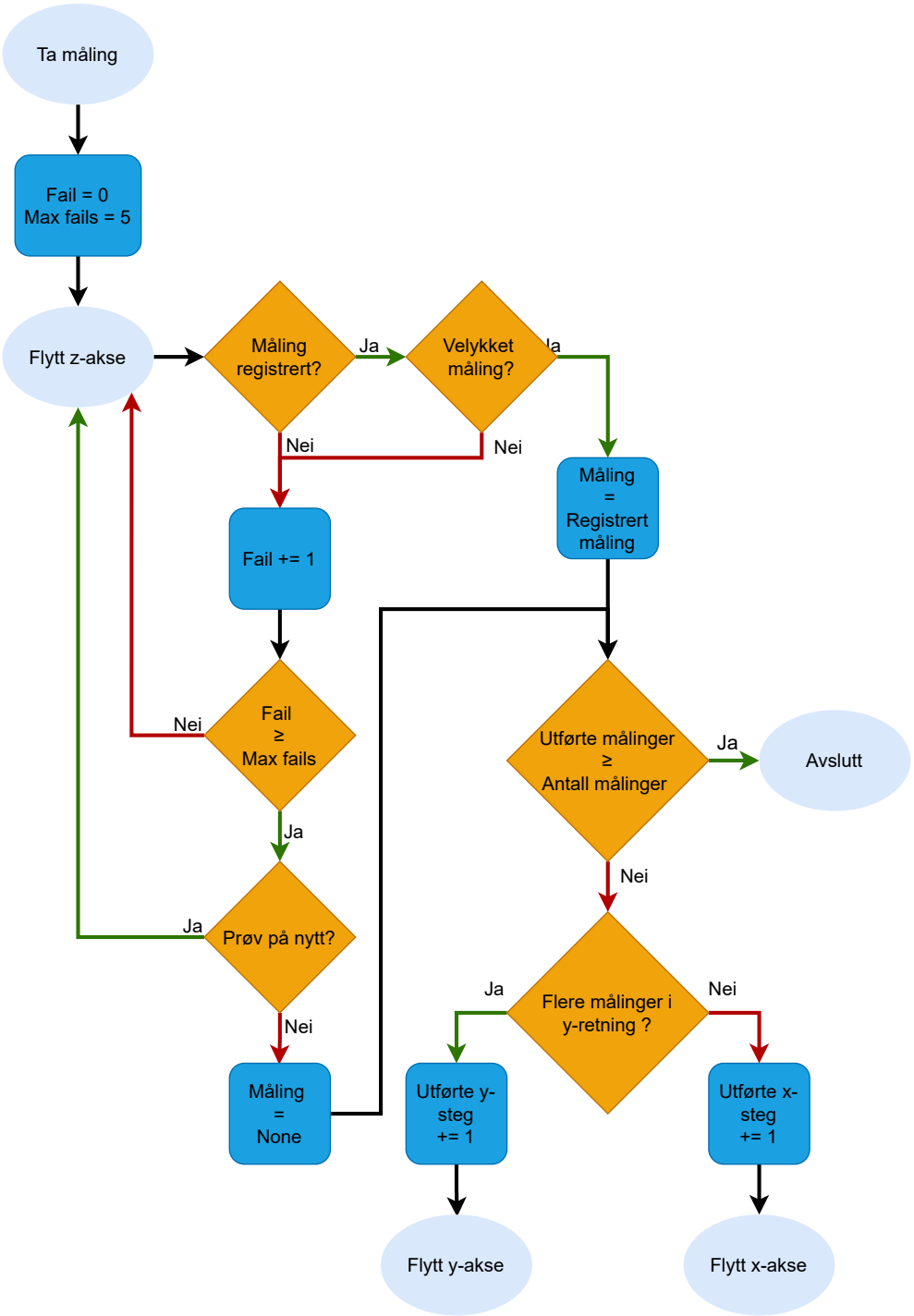
Deretter sjekkes det hvilke pådrag som er det største, med andre ord hvilke operasjoner som vil ha lengst utføringstid og venter til denne utføringstiden er ferdig før roboten offseter seg fra den nye origo.



Figur 3.25: Offseten forskyver hele målesetter slik at målingene havner midt på platen det måles over

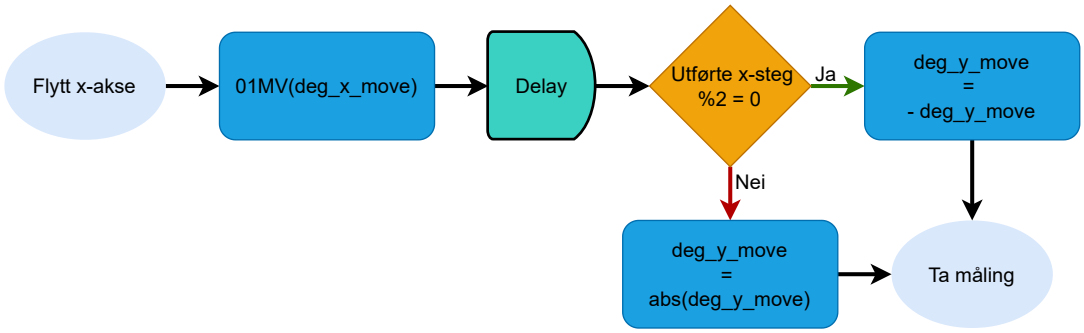
Når en måling skal gjennomføres blir en beskjed sendt for å bevege z-aksen ned, vente en kort tid for så å bevege den opp igjen. Når det er utført, sjekkes det om en måling ble registrert. Hvis det ikke ble oppdaget en ny måling vil den prøve en gang til, i tillegg til å notere at en feil har oppstått. Når en ny måling har blitt registrert blir datapakken som blir mottatt fra Elcometeret gransket for å se at den er som forventet. Dette er for

å forsikre seg om at kun gyldige målinger blir lagt til og registrert. Hvis datapakken ikke er akkurat som det forventede formatet vil programmet også registrere dette som en feil å forsøke og ta en ny måling. Er det fem eller flere feil for et målepunkt vil denne målingen bli registrert som en *None* (en nullverdi), og programmet stanses midlertidig. Det er da mulig å sjekke systemet manuelt, for å se om det er åpenbare feil som kan utbedres. Dette kan for eksempel være kommunikasjonfeil mellom Elcometeret og PC-en, eller om testplaten har flyttet seg, slik at lakken ikke lengre er innenfor måleområdet. Brukeren kan da velge om systemet skal prøve å ta en ny måling på samme plass, eller om programmet skal erstatte målingen med *None*, og fortsette.



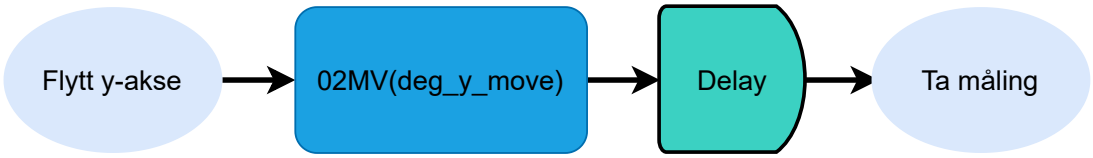
Figur 3.26: Diagrammet viser flyten til programmet når det skal tas en måling

Når målingene er utført uavhengig om det er en gyldig måling eller en *None* sjekker programmet hvor langt den har kommet på y-aksen. Skal det fortsatt tas flere målinger på samme rad, vil funksjonen som beveger y-akse bli kalt opp. Er det ingen flere målinger i y-retning for denne raden vil x-aksen flyttes en steglengde frem. Når x-aksen skal flyttes blir det satt et pådrag som er beregnet ut ifra steglengden mellom hver måling i x-retning



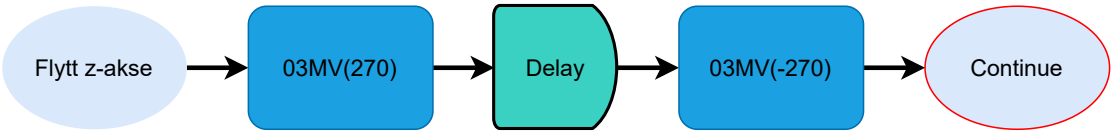
Figur 3.27: Sekvens for forflytte roboten i x-retning og finne ut hvilke retning y-aksen skal bevege seg i

Programmet vil så gjøre en partall-test for å optimalisere kjøringen. Det fungerer ved å se på hvilke steg x-aksen er på. Utførte x-steg begynner på null, som er et partall. Pådraget på y-aksen vil da bli negativ slik at den vil bevege seg innover mot brettet. Når sensoren beveger seg et nytt steg i x-retning vil utførte x-steg bli én, som er et oddetall. Det vil gjøre at pådraget i y-retning blir positivt og roboten vil bevege seg utover. Det resulterer i en mer optimal kjøring.



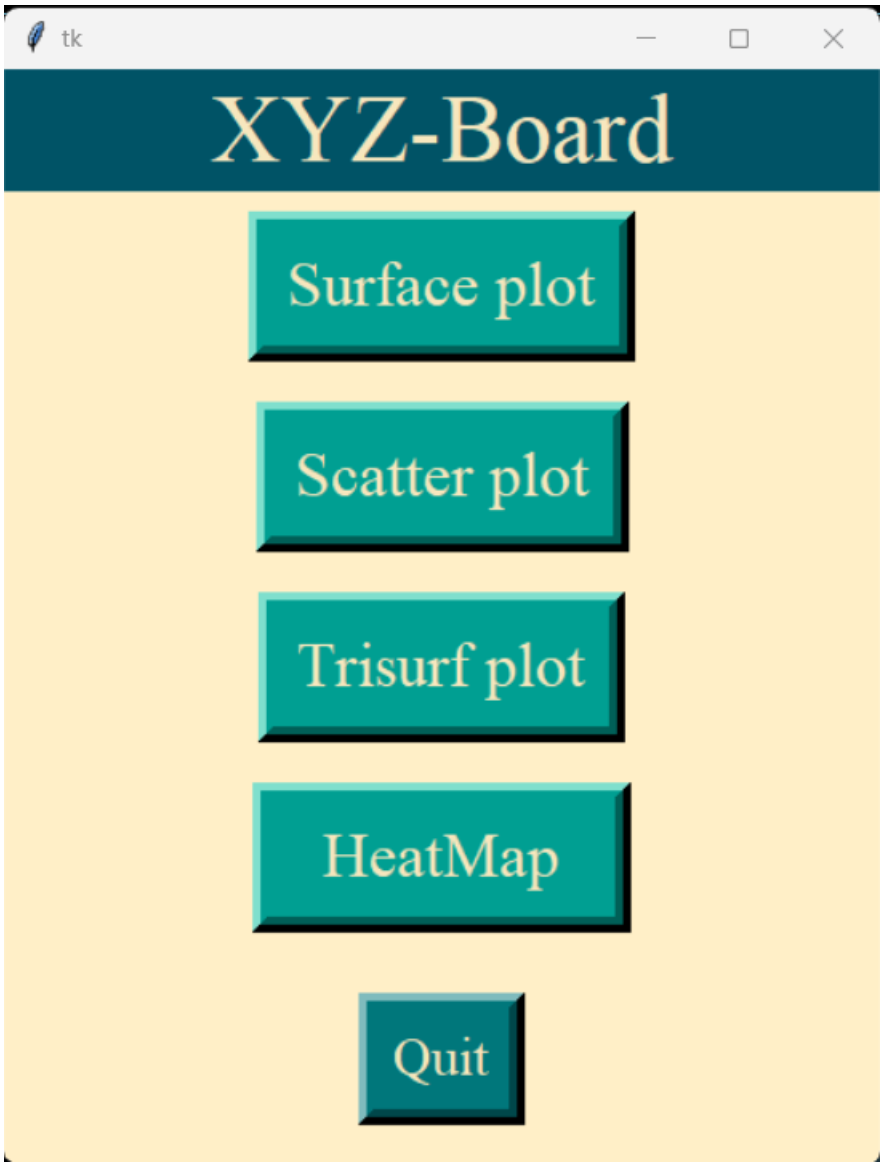
Figur 3.28: Kommandosekvens for å bevege aksen i y-retning

Når bevegelsen i enten x- eller y-retning er utført, blir funksjonen som utfører målinger aktivert på nytt. Flytt z-aksen har kun en funksjon. Den skal flytte proben ned en forhåndsbestemt lengde for så å flytte den opp igjen.



Figur 3.29: Sekvensen med instrukser til den vertikale aksen skal sørge for beveglese opp og ned

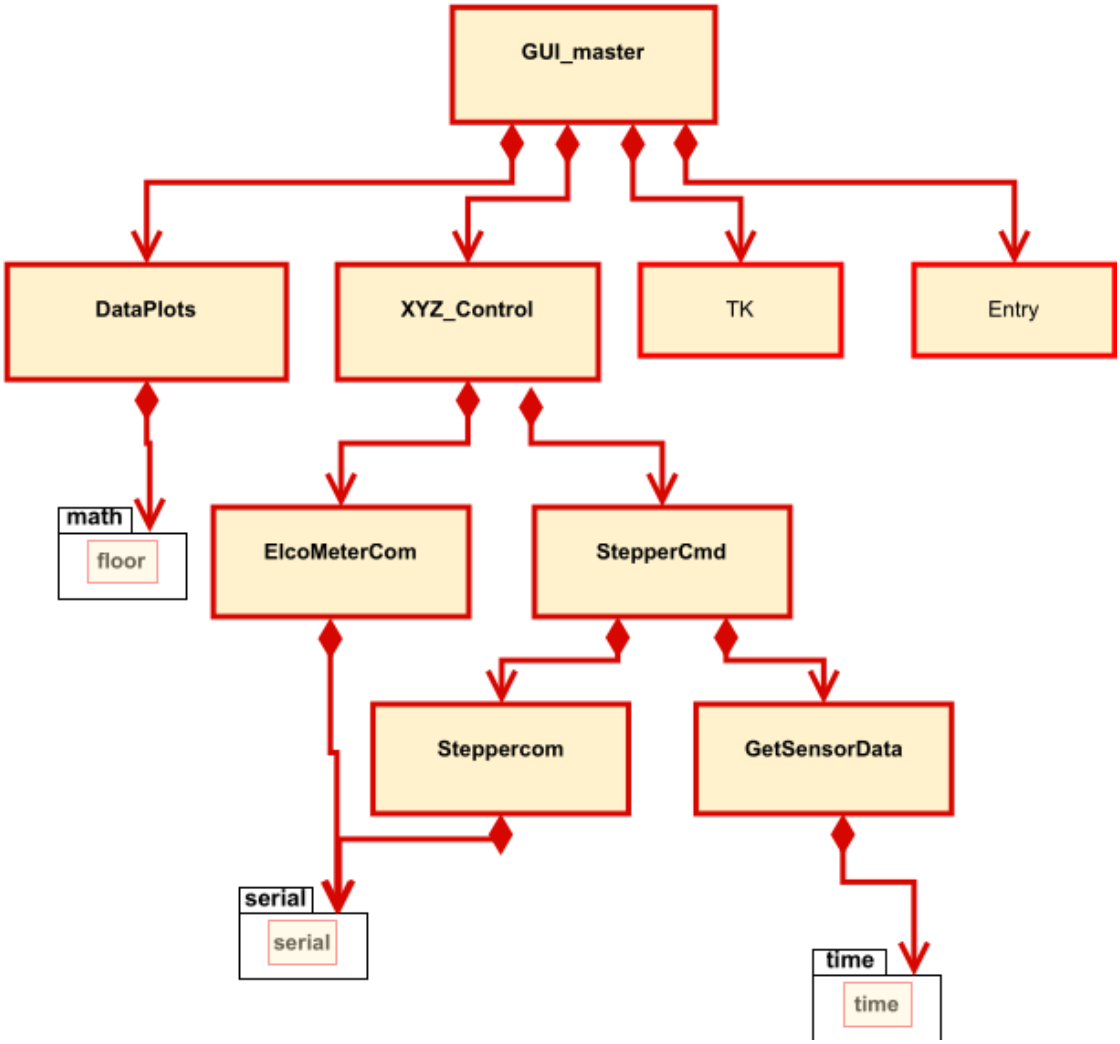
Programmet avsluttes når alle målingene er utført. Det skjer når den siste målingen i y-retning er utført på den siste raden i x-retning. Når dette skjer, stopper robotens bevegelse og motorene blir strømløse. All data som blir sendt fra Elcometeret blir skrevet inn i en Excel-fil, umiddelbart etter programslutt, slik at det kan lagres og benyttes i etterkant av utførelsen. I tillegg åpner et vindu for enkel fremvisning av dataen til brukeren som vist i figur 3.30



Figur 3.30

Når systemet er ferdig med kjøringen er det mulig å presentere lakktykkelsen visuelt med forskjellige metoder. De fleste metodene er 3D-presentasjoner ved bruk av plots som scatter plot, surface plot og trisurf plot. Scatter plot er en fin metode til å visuelt kunne sjekke at spredningen av målingene i måleområdet blir som ønsket. Surface plot og trisurf plot er to forskjellige metoder til å kunne visualisere resultatene i 3D. Det er også implementert en funksjon til å fremvise dataen i et heatmap. Dette heatmappet blir presentert på tre forskjellige måter. Det ene er med rådata, den andre er ved å ta gjennomsnittet av nærliggende data, og den siste er ved å bruke Gaussian Blur på datasettet.

Programmet beskrevet i dette kapittelet består av syv klasser med tilhørende metoder. Klassediagram 3.31 viser klassene og deres relasjon til hverandre. Noen av klassene har assosiasjoner mellom to eller flere klasser. Det gjør det mulig for utvalgte klasser og hente attributter og bruke metoder fra hverandre, for å utføre ønsket programflyt.



Figur 3.31: Klassediagram

Klassene i UML-diagrammet er ikke forklart i dette kapittelet, men det er ikke nødvendig for å forstå programstrukturen. Diagrammet kan likevel være et verktøy for å få en oversikt over systemet. Hvis det er ønskelig å se nærmere på koden som er skrevet kan diagrammet hjelpe til med å se hvordan koden er organisert og hvilke klasser som utfører spesifikke funksjoner i programmet. Det er også vedlagt et mer detaljert klassediagram med alle tilhørende attributter og metoder.

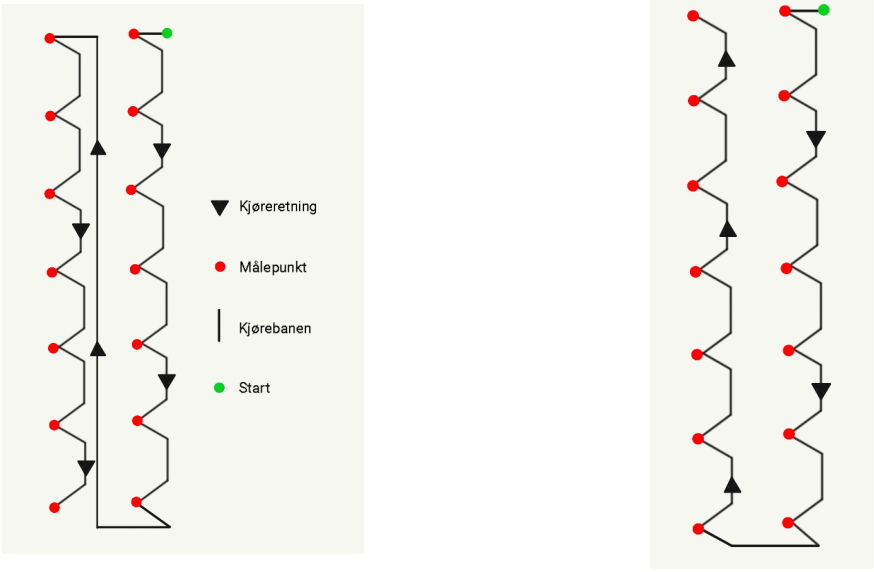
Kapittel 4

Resultater

I dette kapittelet blir resultater av ulike tester og målinger presentert. Ettersom bachelor-oppgaven sitt formål er å lage et system for å ta målinger av lakk. Er det interessant å finne det mest optimale kjøremønster for å effektivisere tidsforbruket. Det blir også utført en kalibrerings prosess for å undersøke at proben virker som angitt i datablad. En typisk kjøring av et måleområde blir også presentert for å vise hvordan systemet kan benyttes, ikke bare i denne oppgaven, men også for ABB videre.

4.1 Resultat av optimalisering av kjøring

For å effektivisere tidsforbruket ved kjøring av systemet er det i utgangspunktet sett på to forskjellige kjøremønster, se fig 4.1a og 4.1b.



(a) ikke optimalt kjøremønster

(b) optimalt kjøremønster

Figur 4.1: test

Ikke optimalt kjøremønster er når måleproben starter i en retning, og tar målinger langs aksen. Når programmet har utført alle målingene langs aksen, starter den på nytt i samme

posisjon som første måling, bortsett fra at den nå har flyttet seg en steglengde videre i motsatt akse. Optimalt kjøremønster derimot, er når programmet er ferdig utført med alle målingene langs den ene aksen, så flytter den sin respektive steglengde i motsatt akse, og tar målinger på vei tilbake til startpunktet. For å teste hvor mye tid som kan reduseres ved å bruke de ulike kjøremønstrene er det utført fire tester med ulikt areal, hvor alle testene er basert på 100 målinger, se 4.1 og 4.2.

Målinger x-retning	Målinger y-retning	x-lengde $[mm]$	y-lengde $[mm]$	steglengde x-retning $[mm]$	steglengde y-retning $[mm]$	Tid $[s]$	mikrostepping	fart $[pps]$
10	10	950	950	95	95	785.98	1/16	10000
10	10	950	100	95	10	321.18	1/16	10000
10	10	100	950	10	95	760.21	1/16	10000
10	10	100	100	10	10	297.27	1/16	10000

Tabell 4.1: Ikke optimalt kjøremønster.

Tabell 4.1 viser resultater, ved kjøremønster vist i 4.1a, av fire forskjellige areal med en mikrostepping på $\frac{1}{16}$. Tilsvarende viser tabell 4.2 resultater av kjøring over samme areal med kjøremønster vist i 4.1b.

Målinger x-retning	Målinger y-retning	x-lengde $[mm]$	y-lengde $[mm]$	steglengde x-retning $[mm]$	steglengde y-retning $[mm]$	Tid $[s]$	mikrostepping	fart $[pps]$
10	10	950	950	95	95	535.34	1/16	10000
10	10	950	100	95	10	290.32	1/16	10000
10	10	100	950	10	95	508.75	1/16	10000
10	10	100	100	10	10	266.78	1/16	10000

Tabell 4.2: Optimalt kjøremønster

Ved å sammenligne tiden systemet bruker på å utføre målingene for hvert område med ulikt kjøremønster, kan tidsbesparingen vises i en ny tabell, se 4.3.

x-lengde $[mm]$	y-lengde $[mm]$	Tid optimal $[s]$	Tid ikke optimal $[s]$	tidsforskjell [%]
950	950	535.34	785.98	-46.8
950	100	290.32	321.18	-10.6
100	950	508.75	760.21	-49.4
100	100	266.78	297.27	-11.4

Tabell 4.3: Tidsbesparing for optimalt og ikke optimalt kjøremønster.

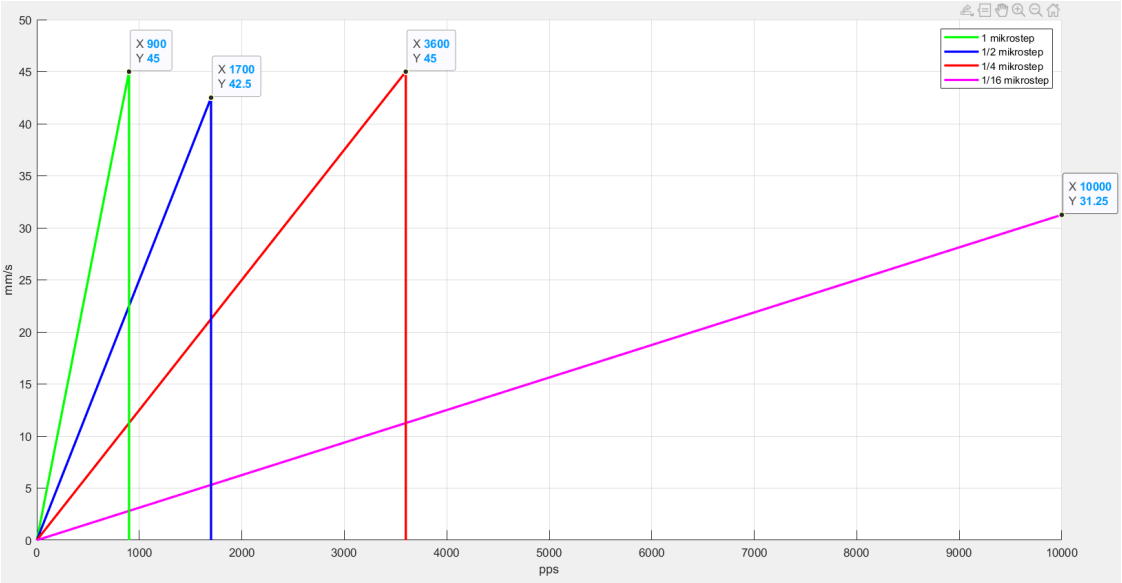
Tabellen over viser tidsforskjellen i prosent for optimalt kjøremønster og ikke optimalt kjøremønster over de ulike områdene. Ettersom det er gjort tester på hvordan ulike kjøremønster påvirker tiden systemet bruker på å utføre målingene, er det interessant å se på om forskjellen på mikrostepping også har en påvirkning på tiden systemet bruker.

Målinger x-retning	Målinger y-retning	x-lengde $[mm]$	y-lengde $[mm]$	steglengde x-retning $[mm]$	steglengde y-retning $[mm]$	Tid $[s]$	mikrostepping	fart $[pps]$
10	10	100	100	10	10	258.38	1/1	900
10	10	100	100	10	10	256.53	1/2	1700
10	10	100	100	10	10	252.88	1/4	3600
10	10	100	100	10	10	297.27	1/16	10000

Tabell 4.4: Optimalt kjøremønster med ulik mikrostepping.

Tabell 4.4 viser resultatene av en test over et område på $100 \cdot 100$ mm med totalt 100 målinger. Her er steppermotorene kjørt til grenseverdiene sine med tanke på pulser per sekund fra motordriveren. Målet med denne testen er å se hvilke mikrostepping som gir best ytelse med tanke på tid, og kunne ta en vurdering på fordeler og ulemper med besparing av tid mot tapt presisjon.

Det er også laget en grafisk fremstilling av hvor fort stegmotorene kan kjøre i mm/s ved de ulike grenseverdiene med tanke på mikrostepping og pulser per sekund fra kontrolleren. Resultatene vises i figur 4.2.



Figur 4.2: Viser korrelasjon mellom pps og forflytning i mm/s for de ulike mikrosteppingene

Ettersom tester for kjøremønster og mikrostepping er utført, ønskes det også å optimalisere ventetiden programmet benytter ved kjøring. Dette gjøres ved stillestående målinger, da programmet benytter en ventetid hver gang proben skal opp og ned for å ta en måling.

antall målinger	forsinkelse ved måling $[s]$	Feil	Feil [%]	Tid $[S]$	forsinkelse i Script $[S]$
64	1.0	0	0	185.89	0.3
64	0.9	0	0	186.56	0.3
64	0.8	0	0	186.08	0.3
64	0.7	0	0	186.33	0.3
64	0.6	0	0	186.55	0.3
64	0.5	0	0	186.42	0.3
64	0.4	16	25	248.03	0.3

Tabell 4.5: kun for stillestående testing

Tabell 4.5 og 4.6 viser resultatene ved testing av samme punkt med ulik ventetid for både måling og script. Resultatene er basert på 64 målinger for hver forsinkelse. Formålet med testen er å finne den ventetiden som gir best ytelse før systemet går for fort slik at proben ikke rekker å registrere en måling.

antall målinger	forsinkelse ved måling $[s]$	Feil	Feil [%]	Tid $[S]$	forsinkelse i Script $[S]$
64	1.0	0	0	153.80	0.2
64	0.9	0	0	138.98	0.2
64	0.8	0	0	142.28	0.2
64	0.7	0	0	153.86	0.2
64	0.6	0	0	155.53	0.2
64	0.5	0	0	148.70	0.2
64	0.4	0	0	149.98	0.2
64	0.3	DNF	DNF	DNF	0.2

Tabell 4.6: kun for stillestående testing

Ved å ta det aritmetiske gjennomsnittet av tiden til de resulterende målingene fra tabell 4.5 og 4.6 er det mulig å estimere hvor mye fortere systemet kan kjøres ved å redusere forsinkelsen i scriptet med 0.1 s. Det aritmetiske gjennomsnittet av kjøringene kan regnes ut med følgende formel.

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n}(x_i) \tag{4.1}$$

$$\bar{x} = \frac{153.80 + 138.98 + 142.28 + 153.86 + 155.53 + 148.70 + 149.98}{7} = 149.02 \tag{4.2}$$

Ligning 4.2 viser gjennomsnittet for kjøringene ved 0.2 s forsinkelse.

$$\bar{x} = \frac{185.89 + 186.56 + 186.08 + 186.33 + 186.55 + 186.42 + 248.03}{7} = 195.12 \tag{4.3}$$

Ligning 4.2 viser gjennomsnittet for kjøringene ved 0.3 s forsinkelse. En kan da bruke disse verdiene til å regne ut tidsbesparingen i prosent når forsinkelsen mellom hver beskjed som blir sendt til kontrollerene reduseres med 0.1 sekunder.

$$\text{Tidsbesparing} = \frac{195.12 - 149.02}{195.12} \cdot 100 = 23.6\% \tag{4.4}$$

Tabell 4.7 viser resultatene av en test hvor det blir kjørt x-antall målinger over et område på $10 \cdot 10$ cm. Tiden det tar systemet å ta disse målingene blir lagt inn i tabellen samt en estimert tid. Tiden per måling blir regnet ut som som et gjennomsnitt av alle målingene se ligning 4.1

$$\text{tid/måling} = \frac{\text{antall målinger}}{\text{totaltid}} \tag{4.5}$$

antall målinger	total tid $[s]$	estimert tid fra modell $[s]$	tid per måling $[s]$
9	27.72	3.03	3.08
16	47.03	2.94	2.94
25	70.13	2.85	2.81
36	98.14	2.77	2.73
49	132.02	2.69	2.69
64	171.03	2.63	2.67
81	212.26	2.59	2.62
100	258.73	2.56	2.59
121	310.56	2.53	2.56
144	366.99	2.52	2.55
196	491.10	2.51	2.51
256	635.98	2.50	2.48
529	1301.66	2.50	2.46

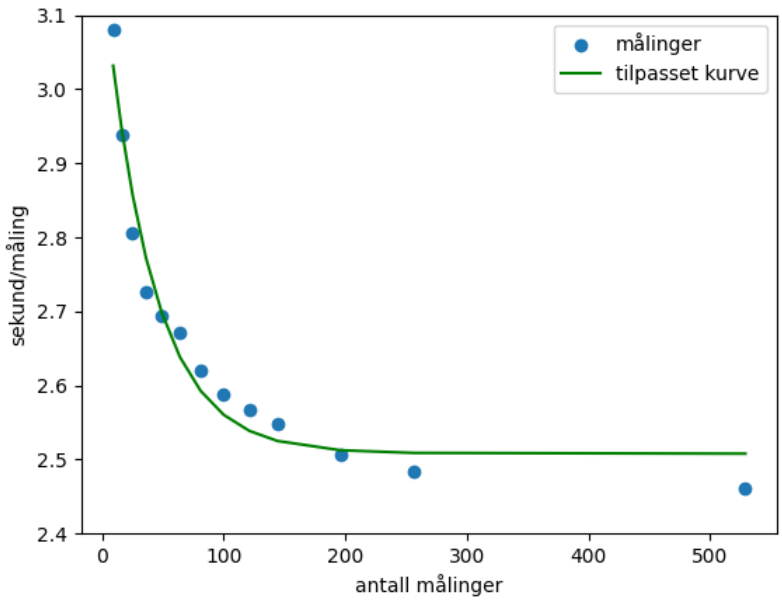
Tabell 4.7: test for et område på $10 \cdot 10$ cm

Ut fra målingene i tabell 4.7 blir dataen plottet og en linje som estimerer tiden per måling. Den estimerte verdien blir regnet ut ved hjelp av ligningen 4.6 her er verdiene a, b og c gitt ut fra curve_fit funksjonen fra pakken scipy.optimize i python og x er antall målinger per datasett. Alt dette blir da plottet inn i grafen 4.3

$$\text{Estimat} = a \cdot e^{-b \cdot x} + c \tag{4.6}$$

Den eksponentielle regresjonslinjen blir da

$$0.6577 \cdot e^{-0.025 * x} + 2.507 \tag{4.7}$$



Figur 4.3: Eksponentiell regresjonslinje for målt tid fra tabell 4.7

4.2 Kalibrering

Presisjonen på proben er kjent fra produsent og er oppgitt til å være ±1%. En rekke tester blir utført for å verifisere at proben har den presisjonen som er oppgitt. Sammen med elcometer følger det med 6 kalibreringsplater på forskjellige tykkelser. Måleinstrumentet må kalibreres til den gitte verdien gjennom en kalibreringsprosess på måleinstrumentet. Denne kalibrering prosessen gjøres ved å trykke på *cal* på instrumentet for så å ta 10 målinger av kalibreringsplaten og til slutt skrive inn hva verdien skal være.
Det tas 100 målinger per kalibreringsplate. For å unngå avvik som kommer av kalibrering, ser vi på standardavviket eller varianskoeffisienten for å vurdere hvor presist proben måler.

Referanse verdi $[\mu m]$	Gjennomsnitt $[\mu m]$	standardavvik	varians	Snitt Feil [%]	varianskoeffesient $CV[\%]$
23.0	24.92	0.04	0.002	8.36	0.16
50.8	51.72	0.089	0.007	1.82	0.17
123.9	124.8	0.16	0.028	0.77	0.13
250.2	251.96	0.087	0.007	0.71	0.03
503	504.64	0.48	0.23	0.32	0.1
960	962.5	0.92	0.86	0.26	0.1

Tabell 4.8: Testing av sensor kalibrert for gitt tykkelsen

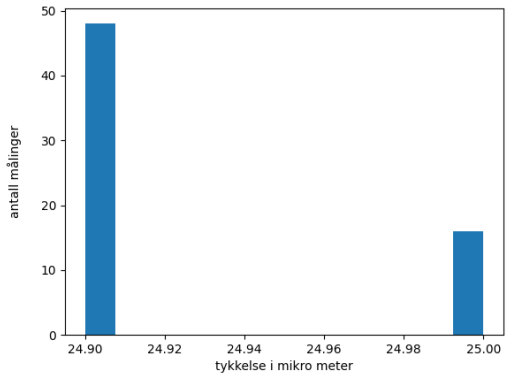
Standardavviket beregnes med formel 4.8 og sier noe om spredningen til målingene i forhold til gjennomsnittet.

$$\sigma = \sqrt{\frac{1}{N} \cdot \sum_{i=1}^{N} (x_i - \bar{x})^2} \tag{4.8}$$

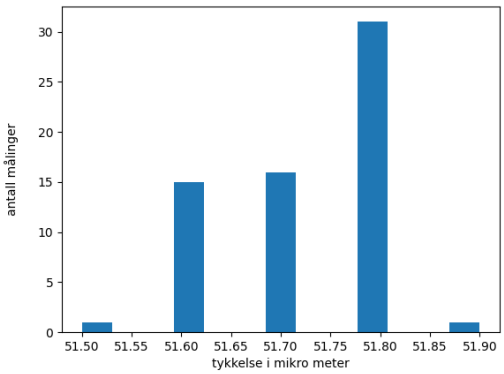
Det kan også være interessant å se på variasjonskoeffisienten eller CV som uttrykker variasjonen i forhold til gjennomsnittet i %. Dette regnes ut slik

$$\frac{\bar{x}}{\sigma} \cdot 100 \tag{4.9}$$

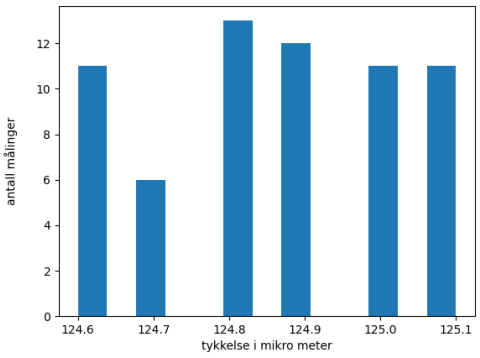
Histogrammer under er brukt for å vise fordelingen av målingene som er tatt i tabell 4.8. Her ser vi hvor mange ganger like målinger er tatt.



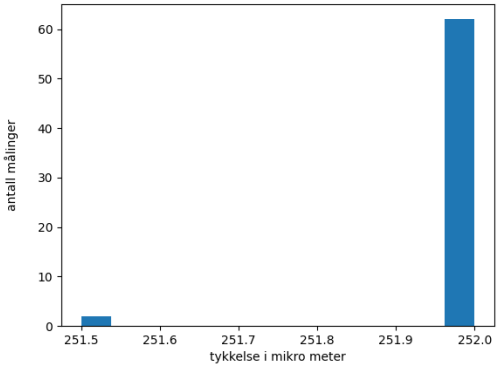
(a) Referanseverdi 23 μm



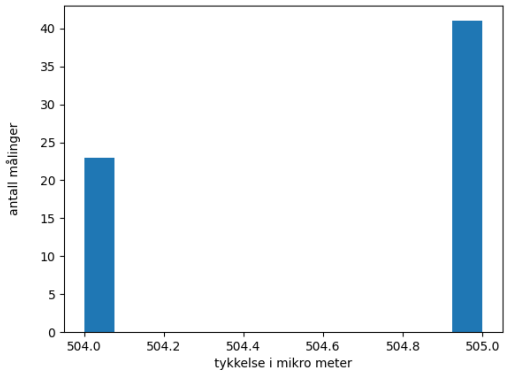
(b) Referanseverdi 50.8 μm



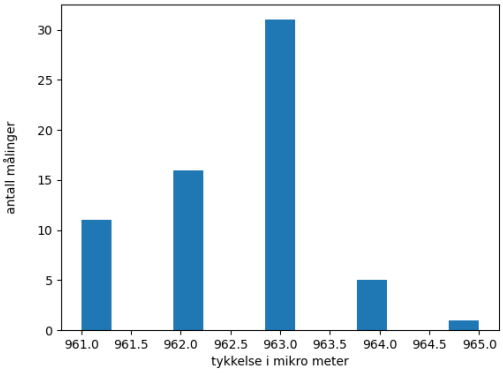
(c) Referanseverdi 123.9 μm



(d) Referanseverdi 250.2 μm

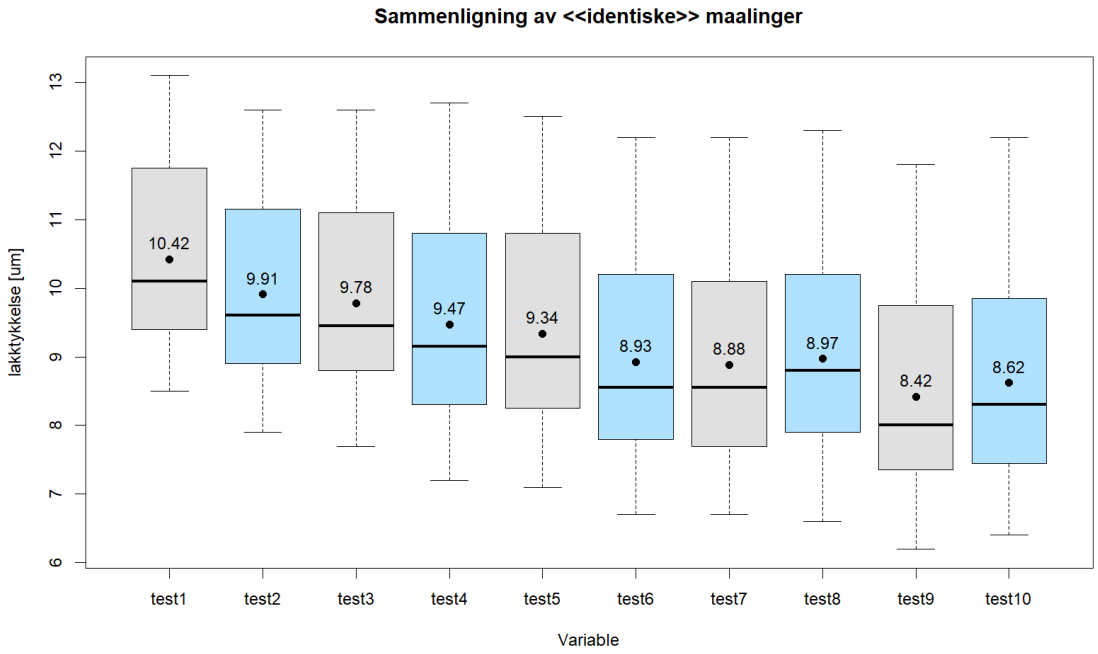


(e) Referanseverdi 503 μm



(f) Referanseverdi 903 μm

For å undersøke stabiliteten av målingene som proben utfører er det kjørt en test over samme punkt med samme antall målinger 10 ganger. Resultatene kan man se i figur 4.5.

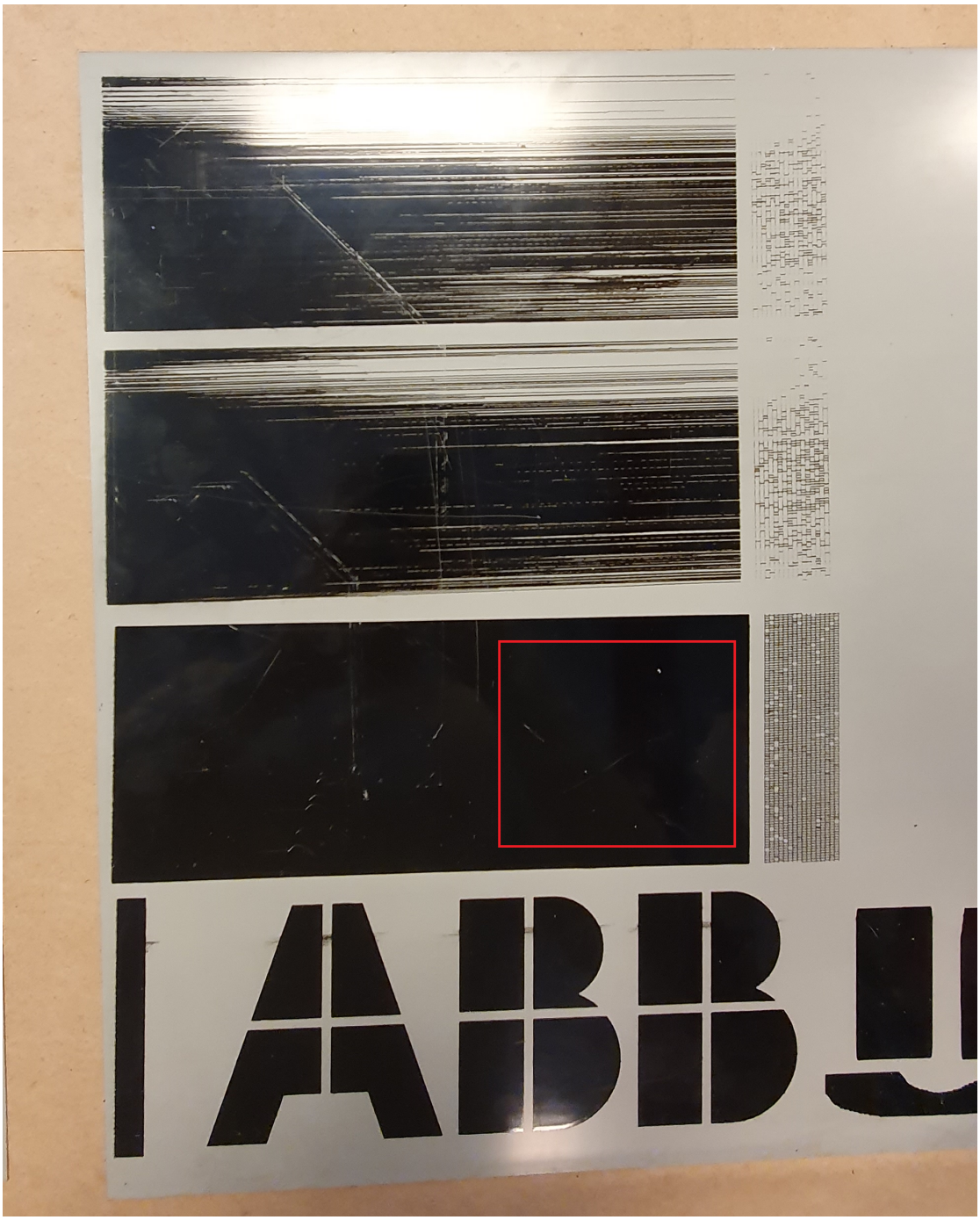


Figur 4.5: Box-plot av tester utført over samme området

Boksen viser intervallet mellom første og tredje kvantil også kalt IQR, (interquartile range). Linjen viser medianen og punktet representerer gjennomsnittet. Topp og bunn verdiene for hvert datasett, altså den stiplede linjen, viser verdien som strekker seg 1.5 ganger fra IQR. Verdier som er utenfor denne verdien anses som utliggere. Hensikten med testen er å undersøke om det oppnås samme resultat eller tilnærmet samme resultat ved de ulike kjøringene.

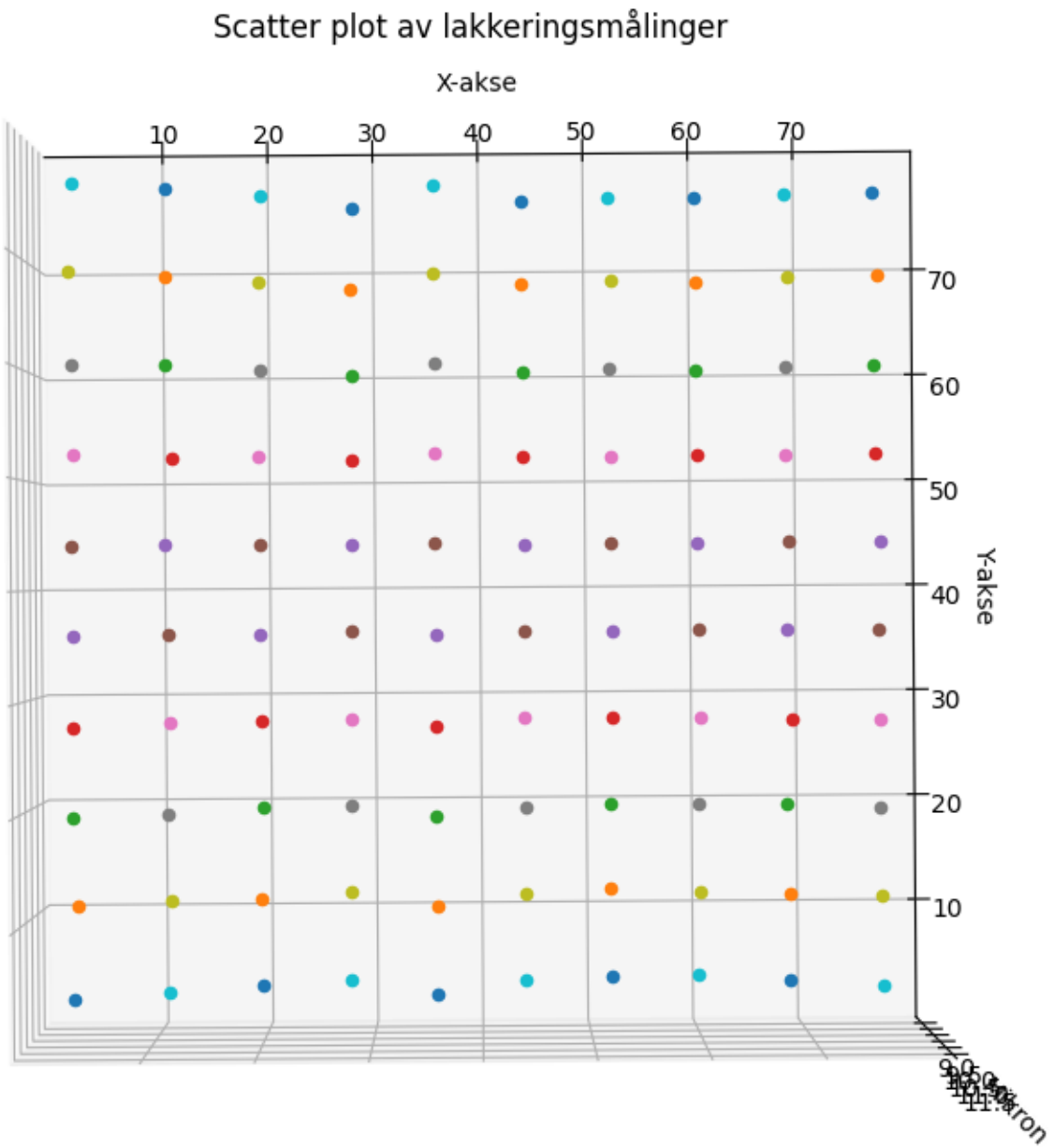
4.3 Presentasjon av en typisk test

Dette delkapittelet tar for seg en typisk test av en testplate. Formålet til delkapittelet er å presentere hvordan en kjøring vanligvis vil kunne presenteres med de ulike funksjonene implementert i systemet. På en lakkert testplate velges et område på $10 \cdot 10$ cm. Det blir tatt 100 målinger jevnt fordelt over det angitte området.



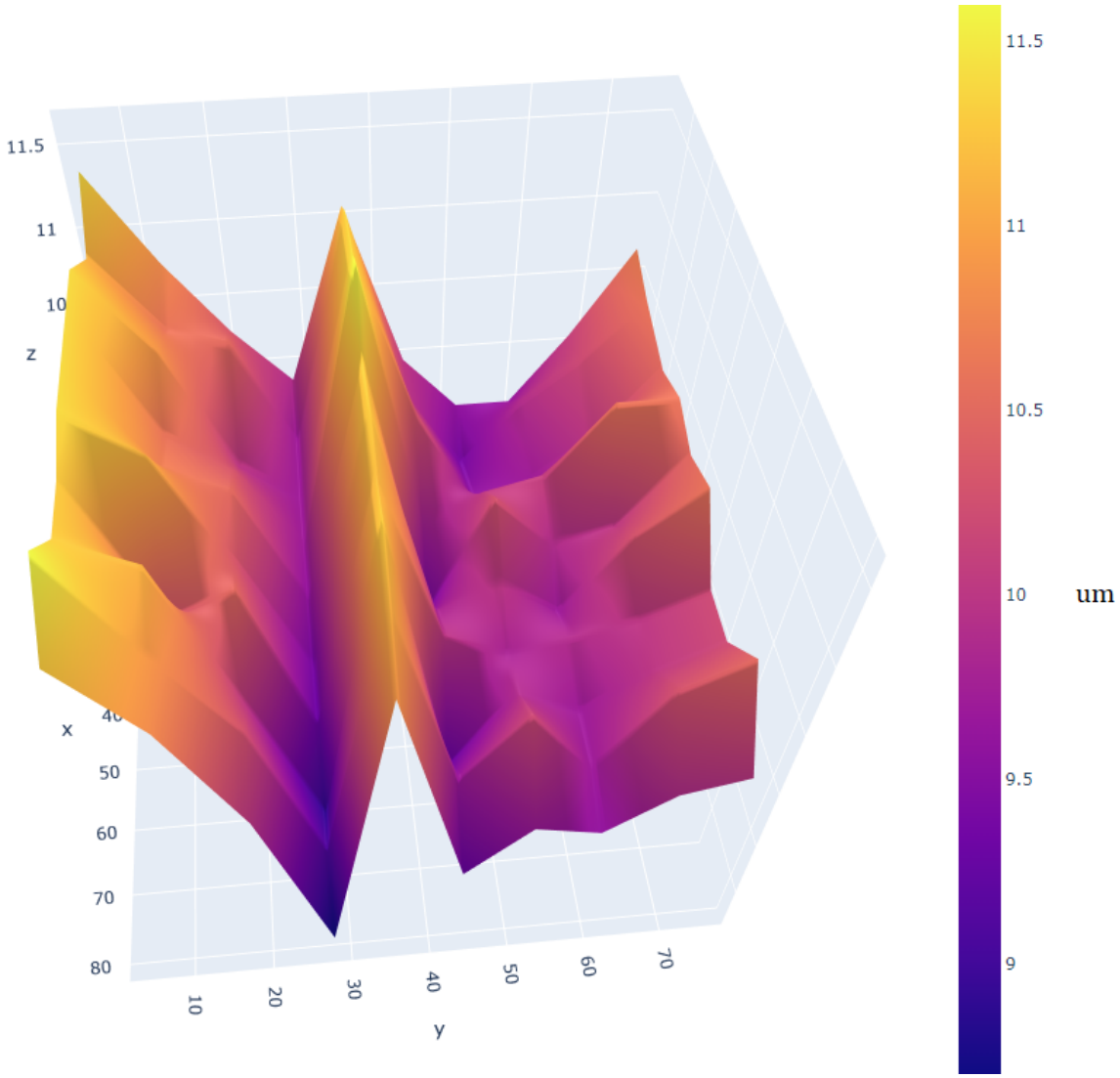
Figur 4.6: Rød firkant definerer måleområde.

Et scatter plott blir generert slik at det er mulig å verifisere at spredningen av målingene er som tiltenkt. Figur 4.7 viser tydelig at spredningen av målepunkter er som ønsket. I figuren ser man måleområdet ovenfra, i et tredimensjonalt rom, derfor kan det se ut som om punktene ikke er helt på linje.



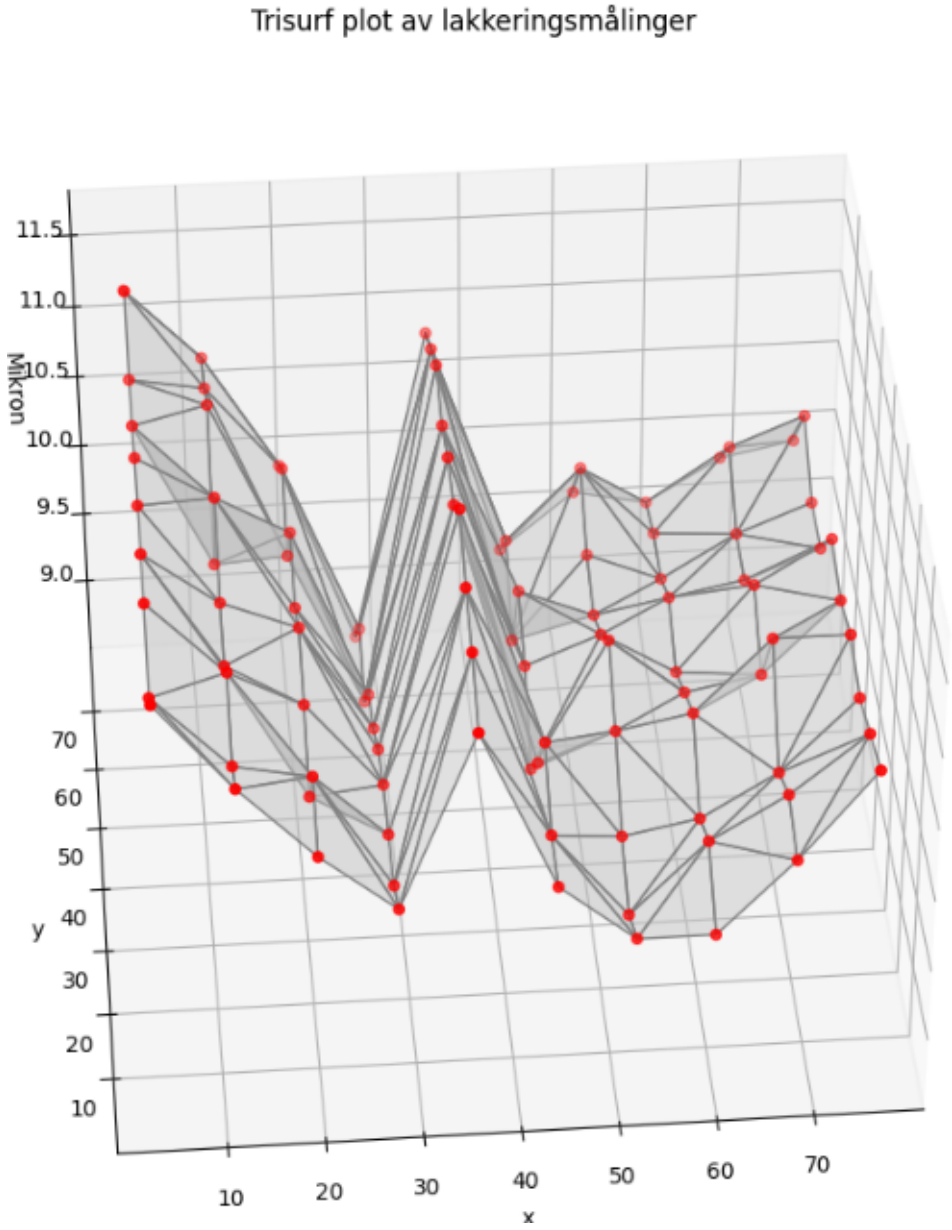
Figur 4.7: Scatter plot av lakkeringsmålinger

Ved å bruke surface plott funksjonen i systemet, kan en visualisere resultatene av målingene i en tredimensjonal representasjon. Dette gjøres ved å representere måleverdiene som høydeverdier på et todimensjonalt rutenett. Den resulterende tredimensjonale overflaten gir en visuell fremstilling av måledataene som kan gjøre det lettere å identifisere mønstre, trender og avvik i datasettet. Se figur 4.8.



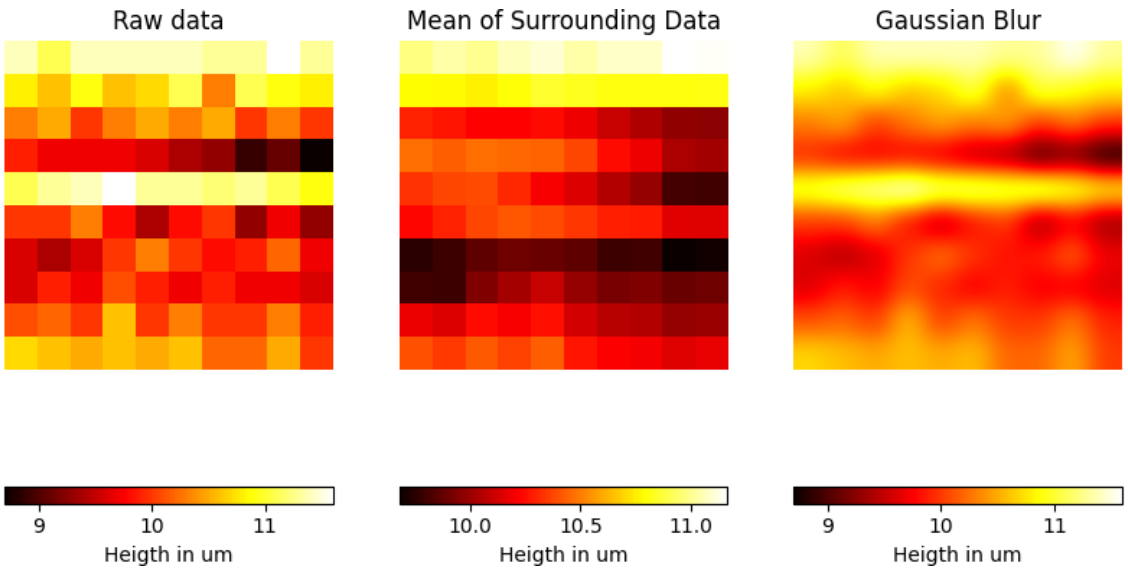
Figur 4.8: Surface plott av målingene

Trisurf plott, se figur 4.9, er en annen metode som kan benyttes til å fremvise dataen tredimensjonalt. Funksjonen til surface plot og trisurf vil være det samme, men en kan velge ut fra plottene etter preferanse med tanke på analyse av dataene visuelt.



Figur 4.9: Trisurf plott av målingene

Ved bruk av *heatmap* funksjonen til systemet genereres det fremvisning av tre ulike *heatmaps* som vist i figur 4.10.



Figur 4.10: Heatmap av området

Rådata kan tidvis være litt vanskelig å analysere, det er derfor utført metoder for å ta gjennomsnittet av nærliggende datapunkter og Gaussian Blur for å gjøre analyser mer brukervennlig.

Ved bruk av den implementerte funksjonen for lagring av data, med tilhørende koordinatsystem, genereres det en Excel-fil. Et utsnitt av Excel-filen for målingene utført i denne testen ser slik ut, se figur 4.11.

	A	B	C	D	E	F
1	Measurement	Value	Unit	X Kor [mm]	Y Kor [mm]	
2	1	11.4	um	4	4	
3	2	11.1	um	4	12	
4	3	11.4	um	4	20	
5	4	11.4	um	4	28	
6	5	11.4	um	4	36	
7	6	11.4	um	4	44	
8	7	11.3	um	4	52	
9	8	11.3	um	4	60	
10	9	11.6	um	4	68	
11	10	11.3	um	4	76	
12	11	10.8	um	12	4	
13	12	10.9	um	12	12	
14	13	11.1	um	12	20	
15	14	10.3	um	12	28	
16	15	11.1	um	12	36	
17	16	10.7	um	12	44	
18	17	10.6	um	12	52	
19	18	10.9	um	12	60	
20	19	10.6	um	12	68	
21	20	10.8	um	12	76	

Figur 4.11: Utsnitt av excel fil generert for målingene

Her kan man se måleverdiene i mikron til alle punktene i kolonne B. Kolonne D og E viser X- og Y koordinater som tilhører sine respektive målenummer som kan leses av i kolonne A. Kolonne B viser måletykkelsen i mikron.

Dataene blir brukt til å lage tabell 4.9. Tabellen viser statistiske verdier for målingene av testplaten.

Referanse verdi $[\mu m]$	Gjennomsnitt $[\mu m]$	standardavvik	varians	snitt error	varianskoeffesient $CV[\%]$
10	10.303	0.6698	0.4486	3.03	6.5

Tabell 4.9: Statistiske verdier for målingene

Kapittel 5

Disuksjon

I denne diskusjonsdelen vil det bli drøftet viktige resultater fra oppgaven, og analysere eventuelle utfordringer som oppsto under utviklingen av systemet og programkoden. Det vil også bli diskutert mulige forbedringer og videre arbeid som kan utføres for å optimalisere systemets funksjonalitet og ytelse.

En av hovedårsakene til å velge å lage et automatisk system er å oppnå effektivisering. Derfor er det viktig å tilrettelegge for effektivisering under konstruksjon og utvikling av programvare til roboten. En måte å effektivisere roboten på er å endre kjøremønsteret som vist i figur 4.1. Selv om dette endrer måten dataene må behandles på, siden halvparten av målingene må transformeres for at matriseformatet skal representere platen riktig, er dette et godt kompromiss ettersom tabell 4.3 viser klare indikasjoner på betydelige tidsbesparelser under kjøring.

Det ble oppnådd så mye som 49.4% reduksjon av kjøretiden hvis den totale forflytningen til y-aksen var lang, men ikke mer enn 11.4% reduksjon hvis y-aksen hadde korte avstander å forflytte seg på. Årsaken til dette er at det er langt flere bevegelser langs y-aksen, ettersom programmet er utviklet for å ta alle målingene i y-aksen før x-aksen skal forflyttes. Dette medfører at ved større avstander i y-aksen, vil effekten av tidsbesparing ved å ikke måtte kjøre helt tilbake før en fortsetter målingen, bli forsterket. Resultatene viser også at ved små avstander er forbedringen mer subtil, men det er fortsatt tegn på forbedring.

I tillegg ble det undersøkt hvordan mikrostepping påvirker utføringstiden til roboten. Ifølge dokumentasjonen skulle stegmotorene kunne rotere med en hastighet som tilsvarer 230 mm/s horisontalt på staget med last, se vedlegg C.2. Etter å ha utført tester på motorene med last, ble det imidlertid oppdaget noen komplikasjoner. Motorene klarte ikke å levere i henhold til spesifikasjonene, og som vist i figur 4.2 var maksimal hastighet som var mulig å oppnå 45 mm/s uavhengig av pps. Dette tilsvarer kun 19.5% av oppgitt hastighet under drift. Hvis motorene ble gitt flere pulser per sekund på de respektive MSF ville de stanse og kun bråke med noen sporadiske steg.

Årsaken til dette fenomenet ble ikke oppdaget, men en annen mulig forklaring kan være at motorkontrolleren ikke leverer nok strøm til motorene. Motoren kan trekke opp til 2 ampere og kontrollerene skal kunne levere opp til 4.2 ampere. Dette burde i utgangspunktet derfor ikke være grunnen til stagneringen. Siden motoren ikke kommer opp i en høyere hastighet enn 45 mm/s og det kun er 1/1 og 1/4 MSF som klarer å levere en slik hastighet, ble det bestemt at roboten skulle kjøres på 1/4 MSF med en pps på 3600, da dette gir det raskeste systemet med best nøyaktighet.

Etter å ha testet repeterbarheten til roboten kan resultatene fra figur 4.5 tyde på at roboten ikke er veldig nøyaktig. Resultatet viser at tykkelsen på lakken tenderer nedover etter flere målinger på det samme området. Dette kan komme av flere årsaker uten at det er mulig å si med sikkerhet hvilke som er tilfelle. En mulighet er at hodet til måleproben trykker ned lakken når den får kontakt, og derfor gjør lakken noe tynnere neste gang den blir målt. En annen mulighet kan være at sensorholderen skader lakken når den blir trykket ned på underlaget og siden holderen er større enn sensoren kan det tenkes at lakken blir skadet før sensoren kommer til det punktet for å måle. Det kan også tenkes at roboten ikke beveger seg presist og nøyaktig slik at sensorhodet ikke er på eksakt samme sted hver gang, og at det er årsaken til ulikheter i forsøkene selv om det ikke i seg selv forklarer hvorfor en observere en synkende trend i lakktykkelse. Lakken som blir testet har ikke blitt videre behandlet, det vil si at den ikke har et lag med klarlakk på toppen. Et slikt lag vil styrke lakkeringen mot slitasje. Det er uvisst om en slik behandling ville hindret den synkende trenden som blir observert i denne testen.

Under konstruksjon- og programmeringsfasen ble det også oppdaget noen utfordringer knyttet til motorkontrollerene. Spenningspotensialet mellom start og jord er på rundt 17V. Dette førte til, som nevnt tidligere, at I/O-enheten konstant leste av høy verdi. Et relé løser dette problemet, men ved bruk av en annen kontroller kunne muligens både kontaktorene og I/O-enheten fjernes. En annen motorkontrollere fra samme produsent SMSD-4.2Modbus kunne vært en bedre kontroller for dette systemet. Fra dokumentasjonen til SMSD-4.2Modbus blir det oppgitt at kontrolleren gir mulighet for direkte avlesning av I/O-portene. Dette ville gjort Moxa-enheten, og dermed også releene overflødig.

I tillegg har kontrolleren flere muligheter for mikrostepping fra 1/1 opp til 1/256. En mikrostepping på 1/256 vil tillate en 14.2 ganger så høy oppløsning enn en mikrostepping på 1/16 og dermed vil systemet også være opptil 1420% mer nøyaktig. Det er allikevel viktig å merke seg at dette bare er en teoretisk økning og at den faktiske økningen vil være avhengig av flere faktorer som for eksempel motorens egenskaper, driverkretsen og mekanikken som brukes i systemet. Det kan likevel antas at systemet ville vært en god del mer nøyaktig med disse kontrollerene. En annen fordel med SMSD-4.2Modbus ville vært mer konsekvente tilbakemeldinger om utførte operasjoner. Dette ville antageligvis også gjort programmet noe mer tidseffektivt. Timerreguleringen som blir brukt har relativt gode estimater for operasjonstid, men timingen kunne vært enda bedre og med tilbakemeldinger i sanntid vil det være mulighet for flere feilhåndteringer og dermed også mulighet for å bygge et mer stabilt software for styring av roboten.

En annen begrensing som skapte utfordringer var minnet på Elcometeret, som kan lagre 10 000 målinger og disse kan bli fordelt på 200 forskjellige batcher. Problemet er bare at en batch kun kan lagre 1000 målinger. Det resulterer i at minne på en batch er fult etter 1000 målinger og Elcometeret klarer da ikke å ta eller sende nye målinger til pc-en. Denne begrensningen gjør at programmet må stoppes etter 1000 målinger for ikke å kjøre uten at noe data blir plukket opp. Løsningen for dette ble å midlertidig pause programmet frem til brukeren tømte minne og avsluttet pausen. Det er ikke en god løsning da dette ikke gjør systemet helautomatisk ved store datamengder. En løsning for å holde systemet autonomt kunne være å installere en smart bryter som er en liten trådløs robot med en funksjon, som er å trykke ned en knapp. Siden minne til Elcometeret kan tømmes med bare å trykke på *clear* knappen kunne dette vært en enkel løsning ved bruk av hardware. En mer elegant løsning kunne vært å endre softwaren til Elcometeret på en slik måte at den ikke lagrer malingen lokalt siden pc-en fanger opp all data og lagrer den.

Kapittel 6

Konklusjon

I denne rapporten er det blitt lagt frem en løsning for ønsket, fra ABB, om et automatisert system for å måle lakktykkelse. Systemet som presenteres er et kartesisk robotsystem som er en fungerende prototype. Det er presentert ulike metoder og analyser for å optimalisere tidsforbruket til systemet, samt stabilitet og kvalitet til målingene.

Resultatene av undersøkelsene utført i denne rapporten viser at valg av spesifikt kjøremønster til systemet er essensielt for å optimalisere tidsforbruket. Analysen viser at det spares på det meste 49.4 % tid ved det optimale kjøremønsteret for et av måleområdene som undersøkes. Det vises også en positiv tidsbesparelse ved å bruke mikrostepping på $\frac{1}{4}$, som tilsvarer en maks hastighet på 45 mm/s. Denne mikrostepping innstillingen blir tenkt å være det mest effektive med tanke på presisjon mot hastighet for formålet til dette systemet. Det er mulig å stille inn motorkontrollerene til $\frac{1}{16}$ for mer nøyaktighet, men dette vil gå utover hastigheten til systemet. Resultatene viser at å redusere tidsforsinkelsen mellom sending av motorstyringskommandoer fra 0.3 sekunder til 0.2 sekunder, tilsvarer det en effektivisering av systemet på 23.6 %.

Selv om undersøkelsene og analysene viser økt effektivitet, er det fortsatt rom for videre utvikling og forbedring av systemet. Ytterligere tester og oppgradering av programvare og maskinvare bør utføres for å gjøre systemet mer stabilt og mer produktivt.

Avslutningsvis så kan oppgaven summeres med at det er utviklet en robot som kan benyttes til å kvalitetsteste lakkering. Systemet har mulighet til å produsere ulike tredimensjonale fremvisninger for analyse, samt datasett som kan brukes til andre formål etter behov.

Bibliografi

[1] ABB. *50 år siden jærbuenes banebrytende oppfinnelse av lakkeringsroboten*. URL: https://new.abb.com/news/no/detail/36667/50-ar-siden-jaerbuenes-banebrytende-oppfinnelse-av-lakkeringsroboten. 01.04.2023.

[2] ABB. *FoU lakkeringsroboter*. URL: https://new.abb.com/no/om-oss/teknologi/fou-lakkeringsroboter. 16.3.2023.

[3] Paul Bjørn Andersen. *automatisering*. URL: https://snl.no/automatisering. 15.3.2023.

[4] V. V. Athani. *Stepper Motors : Fundamentals, Applications And Design*. 1. utg. New Age International, 1997.

[5] Autodesk. *Tinkercad*. URL: https://www.tinkercad.com/. 14.3.2023.

[6] Jan Axelson. *Serial Port Complete*. 2. utg. Lakeview Research, 2007.

[7] Smart Motor Devices. *Programmable Step Motor Controller SMSD-42-RS*. URL: https://smd.ee/manuals/SMSD-4.2RS.pdf. 30.01.2023.

[8] Elcometer. *Coating Thickness Gauges*. URL: https://www.elcometerusa.com/Coating-Thickness-Education/. 17.04.2023.

[9] Elcometer. *Elcometer 355 Coating Thickness Gauge Top Model Operating Instructions*. URL: https://downloads.elcometer.com/PDFs/InstructionManuals/English/355_Top.pdf. 20.3.2023.

[10] Yngve Finnestad. ≪Personlig kommunikasjon≫. Informasjon vedrørende spray vision. 2023.

[11] The Python Software Foundation. *Why was Python created in the first place?* URL: https://docs.python.org/3/faq/general.html#why-was-python-created-in-the-first-place. 16.3.2023.

[12] The R Foundation. *What is R?* URL: https://www.r-project.org/about.html. 10.05.2023.

[13] Dag Håkon Hanssen. *Programmerbare Logiske Styringer*. 4. utg. Fagbokforlaget, 2013.

[14] Geir Martin Haarberg. *Korrosjon*. URL: https://snl.no/korrosjon. 19.04.2023.

[15] Einar Jacobsen. *elektroforese*. URL: https://snl.no/elektroforese. 14.04.2023.

[16] JGraph Ltd. *diagramming for teams*. URL: https://www.drawio.com/. 13.05.2023.

[17] Moxa. *ioLogik E1200 Series*. URL: https://www.moxa.com/getmedia/437fddc2-f39f-4a9a-9851-8a3d1e44cf7e/moxa-iologik-e1200-series-datasheet-v2.3.pdf. accessed: 13.05.2023.

[18] Eirik Rossen & Tom Heine Nätt. *API*. URL: https://snl.no/API. 24.04.2023.

[19] Nils Gundersen, Nils H. Lundberg & Norsk Oljemuseum. *petroleum*. URL: https://snl.no/petroleum. 18.04.2023.

[20] Sven Ore. *Lakk*. URL: https://snl.no/lakk. 14.04.2023.

[21] RS PRO. *RS PRO-PLA TOUGH*. URL: https://docs.rs-online.com/9559/A700000008281484.pdf. 21.02.2023.

[22] Python. *What is Python? Executive Summary*. URL: https://www.python.org/doc/essays/blurb/. 16.3.2023.

[23] Luciano Ramalho. *Fluent Python*. 1. utg. O'Reilly Media, 2015.

[24] Øyvind Grøn & Johannes Skaar. *Elektromagnetisk induksjon*. URL: https://snl.no/elektromagnetisk_induksjon. 17.04.2023.

[25] SNL. *elektroforetisk lakkering*. URL: https://snl.no/elektroforetisk_lakkering. 14.04.2023.

[26] Sven Ore & Aage Stori. *Plast*. URL: https://snl.no/plast. 20.04.2023.

[27] The Gimp Team. *Gimp*. URL: https://www.gimp.org/. 15.3.2023.

[28] Chongqing Umot Technology. *Multi-Axis Xyz Linear CNC Gantry Robot*. URL: https://umottech.en.made-in-china.com/product/SOGtMsBPCuYA/China-Multi-Axis-Xyz-Linear-CNC-Gantry-Robot-3-Axis-Xyz-Linear-Table-Industrial-CNC-Welding-Robot-Robotic-Arm.html. 01.05.2023.

[29] TerraTerm. *Tera Term*. URL: https://ttssh2.osdn.jp/manual/4/en/. 20.1.2023.

Tillegg A

kode

A.1 GUI_master

```
from tkinter import CENTER, E, RAISED, Button, Entry, Frame, Label, Tk, ttk
from tkinter.messagebox import showinfo
import os
import pandas as pd
import time


class GUI_master:

    def __init__(self,master=None):
        self.root=Tk()
        self.color = '#FFEFC7'
        self.txt_list = ['Points', 'X0', 'X1', 'Y0', 'Y1']
        self.entries = []
        self.labels = []
        self.frames = []
        self.root.configure(bg='#FFEFC7')

    def new_window(self):

        #Title
        Label(self.root, text = "User Interface XYZ-Board", fg='#FFEFC7', font=('Times', 33),bg='#005366',anchor=CENTER,width=26).grid(row=0, column=0, columnspan=3)
        Button(self.root,command=self.check_values, bg='#009F92', text='Start', font=('Times', 24),anchor=CENTER, border=8, highlightcolor='green',
                relief=RAISED, activebackground='green').grid(row=6, column=0, columnspan=3, padx= 0, pady= 20)

        # Making the entries and labels
        for i in range(5):
            self.frame = Frame(self.root, padx=5, pady=5, bg='white',highlightbackground = str(self.color), highlightthickness = 2, bd=0)
            self.frame.grid(row=i+1, column=1, pady=5, padx=25)
            label = Label(self.root, bg='#00777B', fg='#FFEFC7', text=self.txt_list[i], font=('Rubik', 24), width=15)
            label.grid(row=(i+1), column=0, pady=5, padx=25)
            entry = Entry(self.frame, width=15, bg = '#FFFCF9', font=('Times 24'), border=5)
            entry.grid(row=i+1, column=0)
            self.entries.append(entry)
            self.frames.append(self.frame)

        for entry in self.entries:
            entry.bind("<Button-1>", self.clear_entry)
        self.root.mainloop()
```

```
40
41      def check_values(self):
42          self.start_values = []
43          i = 0
44          for entry in self.entries:
45              try:
46                  value = float(self.entries[i].get())
47                  if value >= 0:
48                      self.start_values.append(value)
49                      self.frames[i].config(bg='green')
50                  else:
51                      value = None
52                      self.start_values.append(value)
53                      self.frames[i].config(bg='red')
54                      entry.delete(0, 'end')
55                      entry.insert(0, "Invalid Input")
56              except ValueError:
57                  value = None
58                  self.start_values.append(value)
59                  self.frames[i].config(bg='red')
60                  entry.delete(0, 'end')
61                  entry.insert(0, "Invalid Input")
62              i+=1
63          if None in self.start_values:
64              print('All values are not valid. Cant start program')
65          else:
66              self.num_mes = self.start_values[0]
67              self.x_start = self.start_values[1]
68              self.x_stop  = self.start_values[2]
69              self.y_start = self.start_values[3]
70              self.y_stop  = self.start_values[4]
71              self.mm_x_len = int(self.x_stop)-int(self.x_start)
72              self.mm_y_len = int(self.y_stop)-int(self.y_start)
73              print('begin program')
74              time.sleep(1)
75              self.root.destroy()
76              self.program_finish_window()
77
78      def clear_entry(self, event):
79          entry = event.widget
80          if entry.get() == "Invalid Input":
81              entry.delete(0, 'end')
82
83      def program_finish_window(self):
84          self.root = Tk()
85          self.root['bg'] = '#FFEFC7'
86          Label(self.root, text="XYZ-Board", fg='#F7E7BE', font=('Times', 36),
            ↪ bg='#005366', anchor=CENTER, width=16).grid(row=0, column=0, columnspan=3)
87
88          # Define button configurations as a list of tuples
89          button_configs = [
90              ('Surface plot', self.Surface, 3),
91              ('Scatter plot', self.Scatter, 4),
92              ('Trisurf plot', self.Trisurf, 5),
93              ('HeatMap', self.HeatMap, 6),
94              ('Quit', self.root.destroy, 8)
95          ]
96
97          # Create buttons in a for-loop
98          for config in button_configs:
99              b = Button(self.root, text=config[0], fg='#F7E7BE', command=config[1],
                ↪ bg='#009F92', font=('Times', 24), anchor=CENTER, border=8, relief=RAISED)
```

```
            b.grid(column=0, columnspan=3, row=config[2], padx=10, pady=10, ipadx=16)

    def progress(self):
        self.root = Tk()
        self.root['bg'] = '#FFEFC7'
        Label(self.root, text = "XYZ-Board", fg='#F7E7BE', font=('Times',
        ↪  36),bg='#005366',anchor=CENTER, width=16).grid(row=0, column=0, columnspan=3)
        # Button 1
        b1 = Button(self.root, text='Surface plot',fg='#F7E7BE', command=self.Surface,
        ↪  bg='#009F92',font=('Times', 24),anchor=CENTER, border=8, relief=RAISED)
        b1.grid(column=0, columnspan=3, row=3, padx=10, pady=10)

        # Button 2
        b2 = Button(self.root, text='Scatter plot',fg='#F7E7BE', command=self.Scatter,
        ↪  bg='#009F92',font=('Times', 24),anchor=CENTER, border=8, relief=RAISED)
        b2.grid(column=0, columnspan=3, row=4, padx=10, pady=10)

        # Button 3
        b3 = Button(self.root, text='Trisurf plot', fg='#F7E7BE', command=self.Trisurf,
        ↪  bg='#009F92',font=('Times', 24),anchor=CENTER, border=8, relief=RAISED)
        b3.grid(column=0, columnspan=3, row=5, padx=10, pady=10)

        # Button 4
        b4 = Button(self.root, text='HeatMap', fg='#F7E7BE', command=self.HeatMap,
        ↪  bg='#009F92',font=('Times', 24),anchor=CENTER, border=8, relief=RAISED)
        b4.grid(column=0, columnspan=3, row=6, padx=10, pady=10, ipadx=16)

        # Button stopp
        b_stopp = Button(self.root, text='Quit', fg='#F7E7BE', command=self.root.destroy,
        ↪  bg='#00777B',font=('Times', 20),anchor=CENTER, border=8, relief=RAISED,
        ↪  activebackground='#FF0000')
        b_stopp.grid(column=0, columnspan=3, row=8, padx=20, pady=20)


        self.root.mainloop()

    def Surface(self):
        self.DP.Surface(self.XYZ.EMC.list_mes)

    def Scatter(self):
         self.DP.Scatter(self.XYZ.EMC.list_mes)

    def Trisurf(self):
        self.DP.Trisurf(self.XYZ.EMC.list_mes)

    def HeatMap(self):
        self.DP.HeatMap(self.XYZ.EMC.list_mes)

if __name__ == '__main__':

    GUI_master().new_window()
```

A.2 Data_plots

```
from matplotlib.collections import PolyCollection
import numpy as np
import plotly
import plotly.graph_objects as go
import matplotlib.pyplot as plt
```

```
import random
import math
import os
from datetime import datetime
from mpl_toolkits.mplot3d import Axes3D
from scipy.stats import multivariate_normal


class DataPlots:
    def __init__(self, args):
        self.start_verdier = args
        self.num_mes = self.start_verdier[0]
        self.x_start = self.start_verdier[1]
        self.x_stop  = self.start_verdier[2]
        self.y_start = self.start_verdier[3]
        self.y_stop  = self.start_verdier[4]
        self.mm_x_len = int(self.x_stop)-int(self.x_start)
        self.mm_y_len = int(self.y_stop)-int(self.y_start)
        self.Målepunkter()
        self.current_time = datetime.now().strftime('Date%d Time %H-%M')

    def New_folder(self, plot_type):
        if plot_type == 'Surface':
            sub_folder = '/Data/Surface/'
        if plot_type == 'Trisurf':
            sub_folder = '/Data/Trisurf/'
        if plot_type == 'Scatter':
            sub_folder = '/Data/Scatter/'
        if plot_type == 'HeatMap':
            sub_folder = '/Data/HeatMap/'

        try:
            folder_name = f'plot_{datetime.now().strftime("%d-%m-%Y")}'
            self.folder_dir = f'{os.getcwd()}{sub_folder}{folder_name}'
            os.makedirs(self.folder_dir)
        except FileExistsError:
            self.folder_dir = f'{os.getcwd()}{sub_folder}{folder_name}'
            pass

    def Surface(self, args):
        #self.random_z()
        self.Z_List_Value = args
        self.liste_handler_surface()
        grid = np.array(self.Z_List_Value)
        grid = grid.reshape(int(self.x_points),int(self.y_points))
        grid[1::2,:] = grid[1::2, ::-1]
        #print(grid)
        Fig = go.Figure(go.Surface(x=self.x_kordinater,y=self.y_kordinater,z=grid))
        self.New_folder('Surface')
        FileName = (f'{self.folder_dir}/SurfacePlot {self.current_time}.png')
        FileName_html = (f'{self.folder_dir}/SurfacePlot {self.current_time}.html')
        Fig.write_image(FileName)
        plotly.offline.plot(Fig, filename=FileName_html)

    def Trisurf(self, args):
        #self.random_z()
        self.Z_List_Value = args
        print(self.Z_List_Value)
        self.liste_handler_trisurf_scatter()
        x = np.array(self.x_kordinater)
        y = np.array(self.y_kordinater)
        z = np.array(self.Z_List_Value)
```

```
68
69          Fig = plt.figure(figsize=(10,10))
70          Figure = Fig.add_subplot(111, projection='3d')
71          Figure.plot_trisurf(x,y,z, color='white', edgecolor='grey', alpha=0.5)
72          Figure.scatter(x,y,z, c='red')
73          Figure.set_xlabel('x')
74          Figure.set_ylabel('y')
75          Figure.set_zlabel('Mikron')
76          Figure.set_title('Trisurf plot av lakkeringsmålinger')
77          self.New_folder('Trisurf')
78          FileName = (f'{self.folder_dir}/TrisurfPlot {self.current_time}.png')
79          plt.savefig(FileName)
80          plt.show()
81
82      def Scatter(self, args):
83          #self.random_z()
84          self.Z_List_Value = args
85          self.liste_handler_trisurf_scatter()
86          Fig = plt.figure(figsize=(10,10))
87          Figure = Fig.add_subplot(projection='3d')
88          for i in range(len(self.Z_List_Value)):
89              Figure.scatter(self.x_kordinater[i], self.y_kordinater[i],
                ↪  self.Z_List_Value[i])
90          Figure.set_title('Scatter plot av lakkeringsmålinger')
91          Figure.set_xlabel('X-akse')
92          Figure.set_ylabel('Y-akse')
93          Figure.set_zlabel('Mikron')
94          self.New_folder('Scatter')
95          filename = (f'{self.folder_dir}/ScatterPlot {self.current_time}.png')
96          plt.savefig(filename)
97          plt.show()
98
99      def HeatMap(self,args):
100         self.Z_List_Value = args
101         matrix =
            ↪  np.array(self.Z_List_Value).reshape(int(self.x_points),int(self.y_points))
102         matrix[1::2, :] = matrix[1::2, ::-1]
103         matrix.astype(float)
104         mean_matrix = np.zeros_like(matrix)
105         neighborhood_size = 1
106         mean_matrix = np.zeros_like(matrix)
107         for i in range(self.x_points):
108             for j in range(self.y_points):
109                 row_start = max(0, i - neighborhood_size)
110                 row_end = min(self.x_points, i + neighborhood_size + 1)
111                 col_start = max(0, j - neighborhood_size)
112                 col_end = min(self.y_points, j + neighborhood_size + 1)
113                 neighborhood = matrix[row_start:row_end, col_start:col_end]
114                 mean = np.mean(neighborhood)
115                 mean_matrix[i, j] = mean
116         fig = plt.figure(figsize=(10,7))
117         fig.add_subplot(1,3,1)
118         im1 = plt.imshow(matrix, cmap='hot', interpolation='None')
119         plt.axis('off')
120         plt.title('Raw data')
121         plt.colorbar(im1, orientation='horizontal', label='Heigth in um')
122
123         fig.add_subplot(1,3,2)
124         im2 = plt.imshow(mean_matrix, cmap='hot', interpolation='None')
125         plt.colorbar(im2, orientation='horizontal', label='Heigth in um')
126         plt.axis('off')
127         plt.title('Mean of Surrounding Data')
```

```
128
129         fig.add_subplot(1,3,3)
130         im3 = plt.imshow(matrix, cmap='hot', interpolation='Gaussian')
131         plt.colorbar(im3, orientation='horizontal', label='Heigth in um')
132         plt.axis('off')
133         plt.title('Gaussian Blur')
134         fig.show()
135         self.New_folder('HeatMap')
136         FileName = (f'{self.folder_dir}/HeatMap {self.current_time}.png')
137         fig.savefig(f'{FileName}')
138
139     def MesPoints(self):
140         try:
141             percent_x_dir = self.mm_x_len/(self.mm_x_len+self.mm_y_len)
142             percent_y_dir = self.mm_y_len/(self.mm_x_len+self.mm_y_len)
143         except ZeroDivisionError:
144             percent_x_dir = 1
145             percent_y_dir = 1
146         for i in range (1, 10000):
147             self.mes_points = round(percent_x_dir*(i)*percent_y_dir*(i))
148             if (self.mes_points >= self.num_mes):
149                 self.x_points = math.floor(i * percent_x_dir)
150                 self.y_points = math.floor(i * percent_y_dir)
151                 self.mes_points = self.x_points * self.y_points
152                 if (self.mes_points < self.num_mes):
153                     self.mes_points_addx = (self.x_points + 1) * self.y_points
154                     self.mes_points_addy = self.x_points * (self.y_points + 1)
155                     if (self.mes_points_addx > self.num_mes) and (self.mes_points_addy >
                        ↪ self.num_mes):
156                         if self.mes_points_addx < self.mes_points_addy:
157                             self.mes_points = self.mes_points_addx
158                             self.x_points += 1
159                         else:
160                             self.mes_points = self.mes_points_addy
161                             self.y_points += 1
162                     elif self.mes_points_addx > self.num_mes:
163                         self.mes_points = self.mes_points_addx
164                         self.x_points += 1
165                     elif self.mes_points_addy > self.num_mes:
166                         self.mes_points = self.mes_points_addyø
167                         self.y_points += 1
168                     else:
169                         self.mes_points = (self.x_points + 1) * (self.y_points +1)
170                         self.x_points += 1
171                         self.y_points += 1
172
173                 if self.num_mes != self.mes_points:
174                     print("There will be: " + str(self.mes_points) + ' measurements taken
                        ↪ to create a rectangle' + "\n")
175                 else:
176                     print("Measurements = " + str(self.mes_points) + "\n")
177                 print("x points: = " + str(self.x_points) + "\n")
178                 print("y points: = " + str(self.y_points) + "\n")
179                 print('Incremental movement in x direction: ', self.mm_x_len /
                    ↪ self.x_points)
180                 print('Incremental movement in y direction: ', self.mm_y_len /
                    ↪ self.y_points)
181                 self.mm_x_move = self.mm_x_len / self.x_points
182                 self.mm_y_move = self.mm_y_len / self.y_points
183                 self.x_start =  self.mm_x_move / 2
184                 self.y_start =  self.mm_y_move / 2
185                 break
```

```
186
187     def liste_handler_surface(self):
188         self.x_kordinater = []
189         self.y_kordinater = []
190 
191         x_kordinat = 0
192         for i in range(int(self.x_points)):
193             if i == 0:
194                 self.x_kordinater.append(self.x_start)
195             elif i == 1:
196                 self.x_kordinater.append(self.x_start+x_kordinat)
197             else:
198                 self.x_kordinater.append((self.mm_x*i)+self.mm_x)
199             x_kordinat += self.mm_x
200 
201         y_kordinat = 0
202         for j in range(int(self.y_points)):
203             if j == 0:
204                 self.y_kordinater.append(self.y_start)
205             elif j == 1:
206                 self.y_kordinater.append(self.y_start+y_kordinat)
207             else:
208                 self.y_kordinater.append((self.mm_y*j)+self.y_start)
209             y_kordinat += self.mm_y
210 
211         return self.x_kordinater, self.y_kordinater
212 
213     def liste_handler_trisurf_scatter(self):
214         self.x_kordinater = []
215         self.y_kordinater = []
216 
217         x_kordinat = 0
218         teller = 0
219         for i in range(int(self.x_points)):
220             if i == 0:
221                 self.x_kordinater.append(self.x_start)
222             elif i == 1:
223                 self.x_kordinater.append(self.x_start+x_kordinat)
224             else:
225                 self.x_kordinater.append((self.mm_x*i)+self.x_start)
226             for p in range(int(self.y_points-1)):
227                 if i == 0:
228                     self.x_kordinater.append(self.x_start)
229                 elif i == 1:
230                     self.x_kordinater.append(self.x_start+x_kordinat)
231                 else:
232                     self.x_kordinater.append((self.mm_x*i)+self.x_start)
233             x_kordinat += self.mm_x
234         for i in range(int(self.x_points)):
235             teller += 1
236             if teller % 2 != 0:
237                 y_kordinat = 0
238                 for j in range(int(self.y_points)):
239                     if j == 0:
240                         self.y_kordinater.append(self.y_start)
241                     elif j == 1:
242                         self.y_kordinater.append(self.y_start+y_kordinat)
243                     else:
244                         self.y_kordinater.append((self.mm_y*j)+self.y_start)
245                     y_kordinat += self.mm_y
246             else:
247                 for i in range(int(self.y_points)):
```

```
                    if i == 0:
                        y_kordinat = y_kordinat-self.mm_y+self.y_start
                    else:
                        y_kordinat -= self.mm_y
                    self.y_kordinater.append(y_kordinat)


if __name__ == '__main__':

    g = DataPlots()
    #g.Surface()
    #g.Scatter()
    g.Trisurf()
```

A.3 GetSensorData

```
import time
from requests import get
from json import loads


class GetSensorData:
    def __init__(self, address):
        self.address = address
        self.connection_error = None # True if there is no connection for more than 1 s
        self.data_error = None
        self.replay_status_code = None # connection statuses = 200; 404 etc
        self.reply = None # request response
        self.json_data = None # converted response to json
        self.deviceUpTime = None
        self.DI = None
        self.sensor1 = False
        self.sensor2 = False
        self.sensor3 = False
        self.sensor4 = False
        self.sensor5 = False
        self.sensor6 = False
        self.sensor7 = False
        self.sensor8 = False
        self.sensor_list = [self.sensor1, self.sensor2, self.sensor3, self.sensor4,
        ↪  self.sensor5, self.sensor6, self.sensor7, self.sensor8]
        self.index_list = ['01','01','03','03','02','02','01','01']
        self.direction_list = [180, -180, -450, 1530, -180, 180, -180, 180]

    def get_sysInfo(self):
        self.get_data(api_address_extension = "/api/slot/0/sysInfo/device")
        if not self.connection_error:
            self.convert_received_data()
            self.deviceUpTime = self.json_data['sysInfo']['device'][0]['deviceUpTime']

    def get_DI(self):
        self.get_data(api_address_extension="/api/slot/0/io/di")
        if not self.connection_error:
            self.convert_received_data()
            self.DI = self.json_data['io']['di']

    def run(self):
        self.get_sysInfo()  # get device status
        self.get_DI() # get digital inputs

```

```
    def get_data(self, api_address_extension = "/api/slot/0/io/do"):
        try:
            self.reply = get(self.address + api_address_extension,
                             {"rel_rhy": "network"},
                             headers={"Content-Type": "application/json", "Accept":
                             ↪  "vdn.dac.v1"},
                             timeout=0.1,
                            )

            self.replay_status_code = self.reply.status_code
            self.connection_error = False
            self.last_get_time = time.time()
        except:
            self.connection_error = True

    def convert_received_data(self):
        if not self.connection_error:
            try:
                json_blob = loads(self.reply.content.decode('utf-8'))
                self.json_data = json_blob

                self.data_error = False

            except ValueError:
                self.data_error = True
```

A.4 StepperCom

```
import serial
import time


class Steppercom:
    def __init__(self, adress):
        try:
            self.ser = serial.Serial()
            self.ser.baudrate = 9600
            self.ser.port = adress
            self.ser.stopbits = serial.STOPBITS_ONE
            self.ser.parity = serial.PARITY_EVEN
            self.ser.bytesize = serial.EIGHTBITS
            self.ser.timeout = 1.0
            self.ser.open()

            if(self.ser.isOpen()==True):
                print('Open Serial Port {} sucessfully'.format(self.ser.port))
            else:
                print('Open Serial Port {} failed'.format(self.ser.port))
        except:
            print('Open Serial Port exception')

    def ClosePort(self):
        if (self.ser.isOpen()==False):
            pass
        else:
            self.ser.close()

    def OpenPort(self):
        if (self.ser.isOpen()==True):
            pass
```

```
        else:
            self.ser.open()

    def Send_Cmd(self,text=""):#,Terminator="\n"
        try:
            if (self.ser.isOpen()==False):
                print("no port open: ")
            textline= ':' + text + "\r\n"
            self.ser.write(textline.encode('utf-8'))
            #print("sending : ", textline) # uncomment this line to see what's being
                                           # sent to the motordriver
            time.sleep(0.01)
        except Exception as e:
            print("Send Cmd Exception ** "+ repr(e))

    def Read_Respons(self,Terminator="\n"):
        try:
            res = ''
            time.sleep(0.050)
            if (self.ser.in_waiting > 0):
                data_str = self.ser.read(self.ser.in_waiting).decode('ascii')
                #print(data_str, end='') # Uncomment to see whats being sent.
                res += data_str
            if not res.find(Terminator): print ("Timeout : No Terminator character
            ↪ found")
            return res
        except Exception as e:
            print("Read_Respons Exception "+ repr(e))
```

A.5 ElcometerCom

```
import time
import serial


class ElcoMeterCom:
    def __init__(self, serialport_name):
        try:
            print('Open Serial Port')
            self.mes = None
            self.list_mes = ''
            self.list_mes = []
            self.list_all_mes = []
            self.ser = serial.Serial()
            self.ser.baudrate = 9600
            self.ser.port = serialport_name
            self.ser.stopbits = serial.STOPBITS_ONE
            self.ser.parity = serial.PARITY_NONE
            self.ser.bytesize = serial.EIGHTBITS
            self.ser.timeout = 1.0
            self.ser.open()
            self.num_mesX = 10 # Hardcode for testing
            self.num_mesY = 5
            self.mes_num = 1
            self.strike = 0

            if(self.ser.isOpen()==True):
                print('Open Serial Port {} sucessfully'.format(self.ser.port))
            else:
                print('Open Serial Port {} failed'.format(self.ser.port))
```

```
        except:
            print('Open Serial Port exception')

    def ClosePort(self):
        self.ser.close()
        print ('Closing Serial Port')

    def Send_measurement(self):
        try:
            while True:
                if (self.ser.isOpen()==False):
                    print("no port open: ")
                number = self.ser.read(24) # Rawdata
                if len(number) >= 24:
                    self.mes = number.decode('utf-8').strip().split()
                    # Mesurment decoded
                    return True
                time.sleep(0.5)
                return False
        except Exception as e:
            print("Send Cmd Exception  ++"+ repr(e))

    def sucsesfull_mes(self):

        if (len(self.mes) == 3) and ('um' in
        self.mes[2]):# and self.mes[0] == self.mes_num:
            self.list_all_mes.append(self.mes)
            self.list_mes.append(float(self.mes[1]))
            return True
        else:
            print('Measurement error : Somthing is wrong!')
            return False
```

A.6 StepperCommands

```
import os
import math as m
from StepperCom import Steppercom
import time
from GetSensorData import GetSensorData

class StepperCmd:
    def __init__(self, comport):
        ### Variables ###
        self.slaveaddr = '00'
        self.stepperCmdList = ''
        self.stepperCmdList = []
        self.status = ''
        self.angScaling = (200*16)/360  #(steps*microstepperfactor)/360 degrees
        self.revolutions = 0
        ### Stepper variables
        self.stepperangle = 0.0
        self.stepperangleideal = 0
        self.stepperanglereal = 0.0
        self.stepperangleError = 0.0
        self.delay = 0.2
        self.delay_cal = 0.3
        self.delay_move_z = 0.5
        self.kalibrerings_tid = 32
        ### motor driver limits ###
```

```
        self.cmdlimits = {
                            'acc':[-1000, 1000],
                            'speed':[1, 10000],
                            'startspeed':[1, 2000],
                            'angleSteps':[1, 10000000],
                            'pausetime':[1, 100000000]
                        }
        super().__init__()
        self.com = Steppercom(comport) #hardcoded for testing
        self.GSD = GetSensorData("http://192.168.127.254")

    def collision_safty(self): # Moves the motor outside of the sensor range if inside
        while True:
            self.GSD.get_DI()
            for i in range(len(self.GSD.sensor_list)):
                if self.GSD.DI[i+1].get('diStatus') == 1:
                    if self.GSD.sensor_list[i] == True:
                        pass
                    else:
                        time.sleep(self.delay + 0.1)
                        self.setadr(self.GSD.index_list[i])
                        time.sleep(self.delay + 0.1)
                        self.anglego(float(self.GSD.direction_list[i]))
                        self.GSD.sensor_list[i] = True
                elif self.GSD.DI[i+1].get('diStatus') == 0:
                    self.GSD.sensor_list[i] = False

    def calibrate(self): # Calibrates the sensor by moving it to a predefined corner
        self.setadr('00')
        self.pwron()
        time.sleep(self.delay_cal)
        self.speed(10000)
        time.sleep(self.delay_cal)
        adr = ['03','01','02']
        ang = [-2000, 40000, 40000]
        ang2 = [90, -500, -500]
        if True in self.GSD.sensor_list:
            print('Sensor Triggered, repositioning')
            for i in range(len(adr)):
                time.sleep(self.delay_cal)
                self.setadr(adr[i])
                time.sleep(self.delay_cal)
                self.anglego(float(ang2[i]))
                time.sleep(1)

        for i in range(len(adr)):
            time.sleep(self.delay_cal)
            self.setadr(adr[i])
            time.sleep(self.delay_cal)
            self.anglego(float(ang[i]))
            time.sleep(1)
        j = self.kalibrerings_tid # Number of seconds to wait, max = 25
        for i in range(j): # Wait to make sure calibration is complete, not time optimal
            print('Program start in ',j,' seconds')
            j -= 1
            time.sleep(1)
            #os.system("cls")
        if True in self.GSD.sensor_list:
            print('Fault in Calibration')
            for i in range(len(adr)):
                self.setadr(adr[i])
                self.anglego(float(ang2[i]))
```

```
                time.sleep(1)
            self.calibrate()
        print('Calibration Complete')

    def move_x(self, angle):
        time.sleep(self.delay) #Need to wait to ensure all commands are being sent
        self.setadr('01')
        time.sleep(self.delay)
        self.anglego(angle)
        sleep = (abs(angle)/10000)*self.angScaling
        # Wont move on before estimated time of completion
        if sleep <= self.delay:
            sleep = self.delay
        time.sleep(sleep)

    def move_y(self, angle):
        time.sleep(self.delay)
        self.setadr('02')
        time.sleep(self.delay)
        self.anglego(angle)
        sleep = (abs(angle)/10000)*self.angScaling
        if sleep <= self.delay:
            sleep = self.delay
        time.sleep(sleep)

    def move_z(self, angle):
        time.sleep(self.delay)
        self.setadr('03')
        time.sleep(self.delay)
        self.anglego(angle)
        sleep = (abs(angle)/10000)*self.angScaling
        if sleep <= self.delay:
            sleep = self.delay
        time.sleep(self.delay_move_z+sleep)
        self.anglego(-angle)

    def get_stepperCmdList(self):
        'Current cmd list'
        return self.stepperCmdList

    def setadr(self, args):
        self.slaveaddr = args

    def stop(self):
        'start/stop motor'
        self.stepperCmdList = [self.slaveaddr + 'ST']
        self.com.Send_Cmd(self.get_stepperCmdList()[0])
        self.status = 'Starting or stopping motor'

    def pwron(self):
        'Motor power on'
        self.stepperCmdList = [self.slaveaddr + 'EN']
        self.com.Send_Cmd(self.get_stepperCmdList()[0])
        self.status = 'Motor power on'

    def pwroff(self):
        'Motor power off'
        self.stepperCmdList = [self.slaveaddr + 'DS']
        self.com.Send_Cmd(self.get_stepperCmdList()[0])
        self.status = 'Motor power off'

    def speed(self, value):
```

```
149         'Set motor speed'
150         if self.cmdlimits['speed'][0] <= value <= self.cmdlimits['speed'][1]:
            ↪ #Checking if given acc value is within limits
151             self.stepperCmdList = [self.slaveaddr + 'SD' + str(value)]
152             self.com.Send_Cmd(self.get_stepperCmdList()[0])
153             self.status = 'Motor rotation speed set to: ' + str(value)
154         else:
155             self.status = 'Given speed value is out of range. Valid range is ' +
                ↪ str(self.cmdlimits['speed'][0]) + ' to ' +
                ↪ str(self.cmdlimits['speed'][1])
156
157     def anglego(self, value):
158         'Run stepper motor alfa degrees relative to current position'
159         if value > 0:
160             if self.cmdlimits['angleSteps'][0] <= value*self.angScaling <=
                ↪ self.cmdlimits['angleSteps'][1]:
161                 self.stepperCmdList =  [
162                     self.slaveaddr + 'DL',  # Forward motion
163                     self.slaveaddr + 'MV' + str(round((value -
                        ↪ self.stepperangleError)*self.angScaling)),
                        ↪ # Move numbers of steps
164                                         ]
165                 #print(self.stepperCmdList)
166                 for command in self.get_stepperCmdList():
167                     self.com.Send_Cmd(command)
168                     time.sleep(0.05)
169                     # If sent too fast, python will try to send data
170                     # faster that the baudrate?
171                 if self.slaveaddr == '03':
172                     self.stepperanglereal = (round((value -
                        ↪ self.stepperangleError)*self.angScaling)/self.angScaling)
                        ↪ self.stepperangleideal += value
173                     self.stepperangle += self.stepperanglereal
174                     self.stepperangleError = self.stepperangle - self.stepperangleideal
175                     #print('angle error 1',self.stepperangleError)
176
177
178         elif value < 0:
179             if self.cmdlimits['angleSteps'][0] <= abs(value)*self.angScaling <=
                ↪ self.cmdlimits['angleSteps'][1]:
180                 self.stepperCmdList =  [
181                     self.slaveaddr + 'DR',
182                     self.slaveaddr + 'MV' + str(round(abs(value -
                        ↪ self.stepperangleError)*self.angScaling)),
183                                         ]
184                 for command in self.get_stepperCmdList():
185                     self.com.Send_Cmd(command)
186                     time.sleep(0.05)
                        ↪ #If sent too fast, python will try to send data faster that
187                                         #the baudrate?
188                 if self.slaveaddr == '03':
189                     self.stepperanglereal = (round(value -
                        ↪ self.stepperangleError)*self.angScaling)/self.angScaling
190                     # keeps track of the real angle after quantization
191                     # error from the stepper motor resolution.
192                     self.stepperangleideal += value
193                     self.stepperangle += self.stepperanglereal
194                     self.stepperangleError = self.stepperangle - self.stepperangleideal
195         else:
196             self.stepperCmdList = []
197             self.status = 'Really? 0 degrees?'
```

A.7 XYZ_control

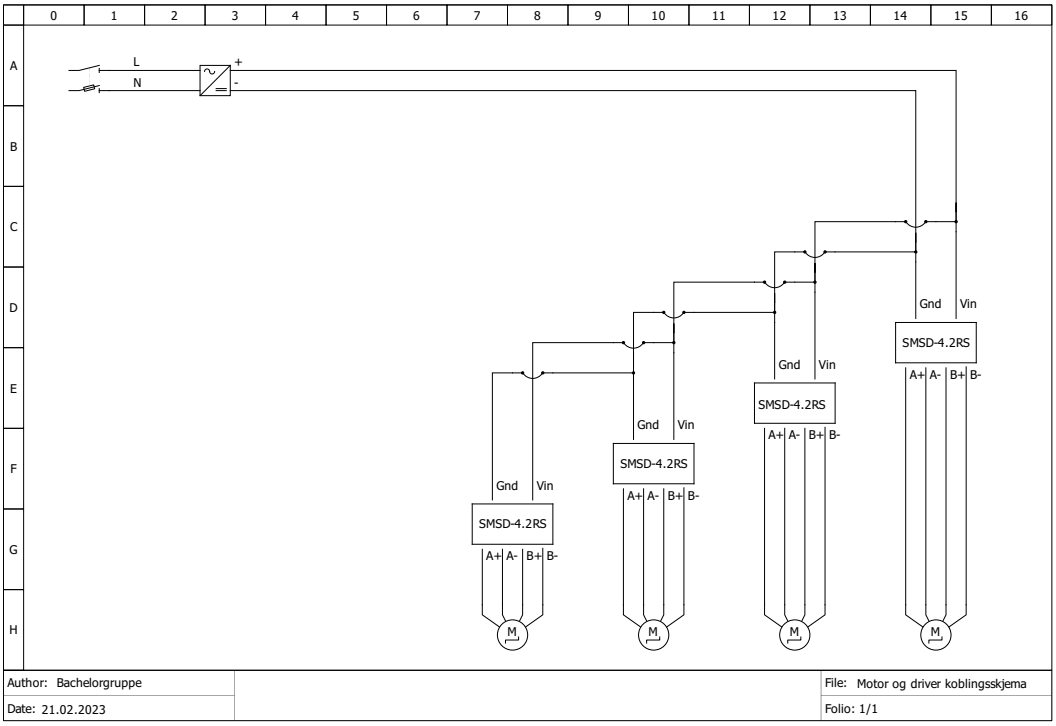
```
from StepperCommands import StepperCmd
from ElcoMeterCom import ElcoMeterCom
import time
import threading as multi
import csv
import numpy as np
import xlsxwriter
import os
from datetime import datetime
import math
import serial.tools.list_ports
from tkinter import *
import tkinter.messagebox as tkm
import serial


class XYZ_Control():
    def __init__(self, args):
        self.comport_provider()
        self.cmd = StepperCmd(self.stepper_comport)
        self.EMC = ElcoMeterCom(self.elcometer_comport)
        multi.Thread(target=self.EMC.Send_measurement, daemon=True).start()
        multi.Thread(target=self.cmd.collision_safty, daemon=True).start()

        self.mes = 0
        self.mesX = args[0]
        self.mesY = args[1]
        self.angle_x = -args[2]*36 # mm to deg
        self.angle_y = args[3]*36
        self.x_start = -args[4]*36
        self.y_start = -args[5]*36
        self.angle_z = 270
        self.delay = 0.2
        self.feiliste = ['NAN', 'NAN', 'NAN']

    def take_measurement(self):
        strike = 0
        while True:
            self.cmd.move_z(self.angle_z)
            if self.EMC.Send_measurement():   # Check if measurement is recived
                if self.EMC.sucsesfull_mes(): # Check if measurement is as expected
                    if int(self.EMC.mes[0]) >= 990:
                        top = Tk()
                        top.geometry('0x0')
                        tkm.showwarning(title='Warning',message='Memory is full \n clear
                        ↪  cache on Elcometer')
                        top.destroy()
                    break
            strike += 1
            print('Strike ', strike, 'try again')
            if strike >= 5:
                top = Tk()
                top.geometry('0x0')
                if tkm.askretrycancel(title='Warning',message=('5 strikes in a row\n'
                                      + 'check Elcometer for fault (clear and/or enter
                                      ↪  may fix the problem)\n'
                                      + 'If there is a metal plate under the sensor press
                                      ↪  retry')):
                    top.destroy()
                    strike = 0
```

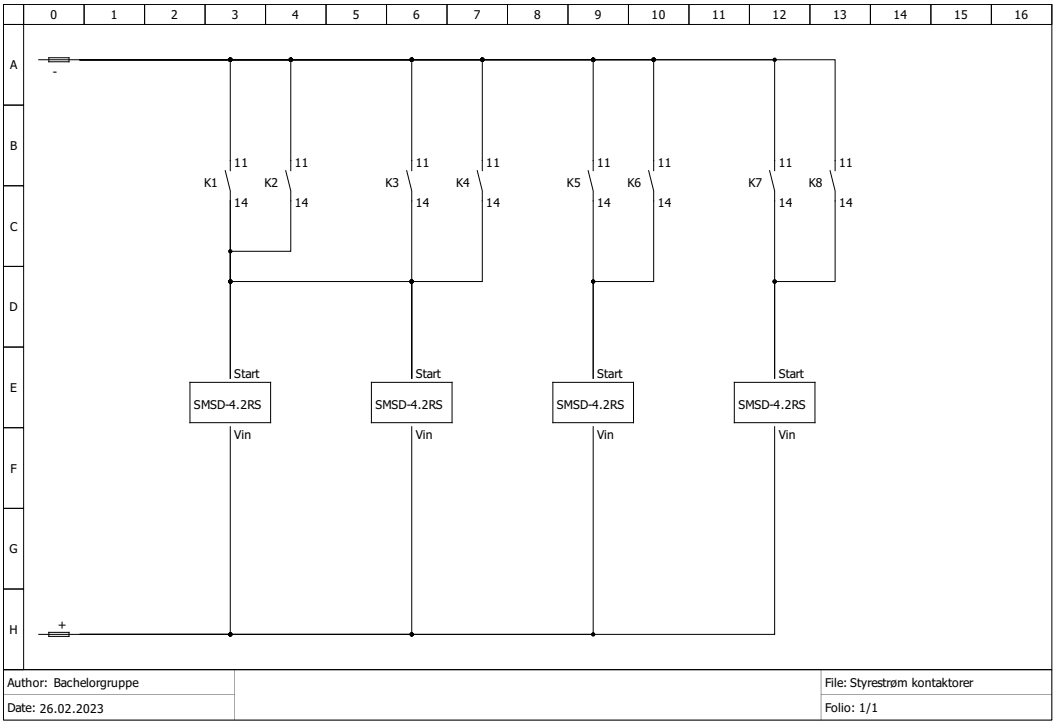
```
                    self.take_measurement()

                else:
                    self.EMC.list_mes.append(np.nan)
                    self.EMC.list_all_mes.append(self.feiliste)
                    print('Measurement error : No measurement can be taken, adding NONE and moving on!')
                    top.destroy()
                    strike = 0
                    break
            time.sleep(self.delay)

    def start_position(self):
        print('Moving to start position')
        self.cmd.setadr('01')
        time.sleep(self.delay)
        self.cmd.anglego(self.x_start)
        time.sleep(self.delay)
        self.cmd.setadr('02')
        time.sleep(self.delay)
        self.cmd.anglego(self.y_start)
        if abs(self.x_start) >= abs(self.y_start):
            angle = self.x_start
        else:
            angle = self.y_start
        sleep = (abs(angle)/10000)*((200*16)/360) # Wont move on before estimated time of completion
        if sleep <= self.delay:
            sleep = self.delay
        #time.sleep(sleep)
        j = math.ceil(abs(angle)/10000)*((200*16)/360) # Wont move on before estimated time of completion
        for i in range(int(j)): # Wait to make sure calibration is complete, not time optimal
            print('Program start in ',j,' seconds')
            j -= 1
            time.sleep(1)
            #os.system("cls")
        print('Starting program')

    def main_control(self):
        self.cmd.calibrate()

        if self.x_start != 0 or self.y_start != 0:
            self.start_position()
        else:
            pass
        time.sleep(self.delay)
        self.cmd.move_x(self.angle_x/2)
        time.sleep(self.delay)
        self.cmd.move_y(-self.angle_y/2)
        time.sleep(self.delay)
        t = time.time()
        for i in range(self.mesX):
            if i > 0:
                self.cmd.move_x(self.angle_x)
                #self.cmd.move_y((abs(self.angle_y*self.mesY)))
                self.take_measurement()
            else:
                self.take_measurement()
            if (i % 2) == 0:
                self.angle_y = -self.angle_y
```

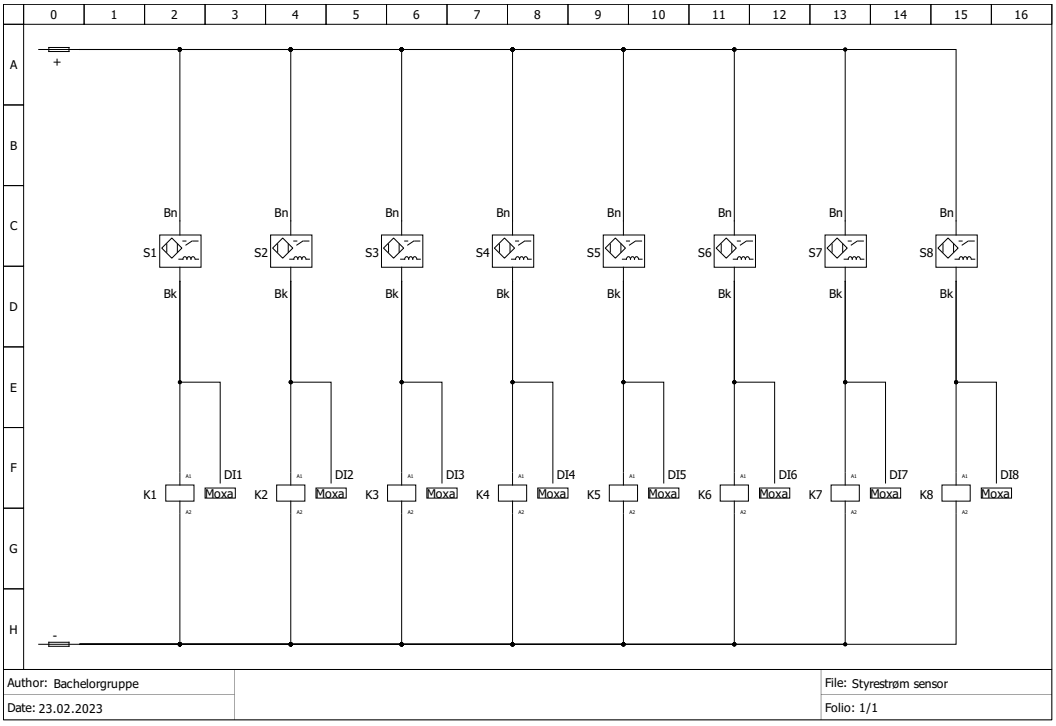
```
116
117             elif (i % 2) != 0:
118                 self.angle_y = abs(self.angle_y)
119
120             for j in range(self.mesY-1):
121                 self.cmd.move_y(self.angle_y)
122                 self.take_measurement()
123             print('runde', i+1, 'ferdig')
124
125         print(time.time() - t)
126         time.sleep(self.delay)
127         self.cmd.setadr('00')
128         time.sleep(self.delay)
129         self.cmd.stop()
130         time.sleep(self.delay)
131         self.cmd.pwroff()
132
133     def comport_provider(self):
134         ports = list(serial.tools.list_ports.comports())
135         for port in ports:
136             if 'Prolific USB-to-Serial Comm Port' in port.description:
137                 self.elcometer_comport = port.device
138             elif 'USB-SERIAL CH340' in port.description:
139                 self.stepper_comport = port.device
140
141
142 if __name__ == '__main__':
143     XYZ_Control()
```

Tillegg B

Koblingsskjema




0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
A B C D E F G H
-
+
K1 K2 K3 K4 K5 K6 K7 K8
11 14
Start
SMSD-4.2RS
Vin
Author: Bachelorgruppe
Date: 26.02.2023
File: Styrestrøm kontaktorer
Folio: 1/1


0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
A B C D E F G H
+
-
Bn
Bk
S1 S2 S3 S4 S5 S6 S7 S8
K1 K2 K3 K4 K5 K6 K7 K8
A1
A2
DI1 DI2 DI3 DI4 DI5 DI6 DI7 DI8
Moxa
Author: Bachelorgruppe
Date: 23.02.2023
File: Styrestrøm sensor
Folio: 1/1

Tillegg C

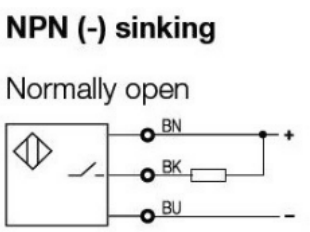
Dokumentasjon

Square Type Inductive Proximity Sensor

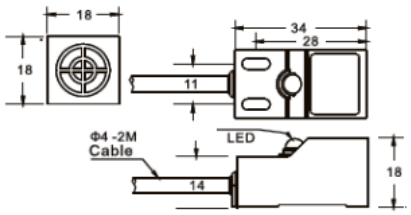
SN04-N

Product Characteristic	
-low cost & stable detection	
-easy mounting	
-water proof and dust proof	
-visible LED	
Electrical Data	
Output	NPN NO
OperatingVoltage	12~24Vdc
Current Consumption	<8mA
Reverse polarity protection	YES
Output	
Function	Normally Open
Load Current	200mA max.
Frequency	600Hz
Sensing Range	
Rated Distance	4mm
Recommended Distance	3.2mm
Hysteresis	10% of Sensing Range
Environment Ratings	
Ambient Temperature	-20~+70°C
Protection Degree	IP67
Material	
Housing material	ABS/PE
Wire	Ø3 x 2M /3 wires

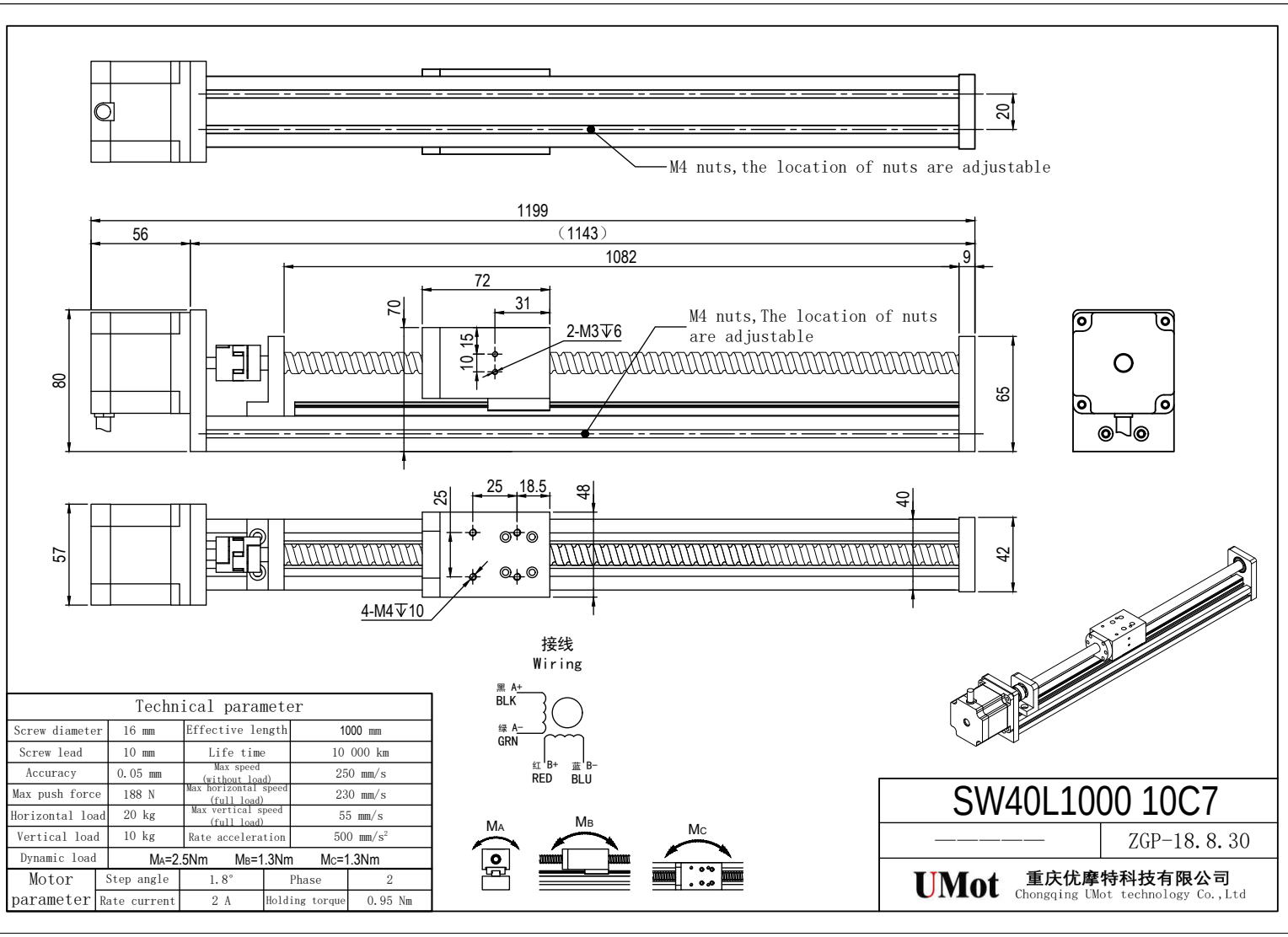
Wiring



Dimension

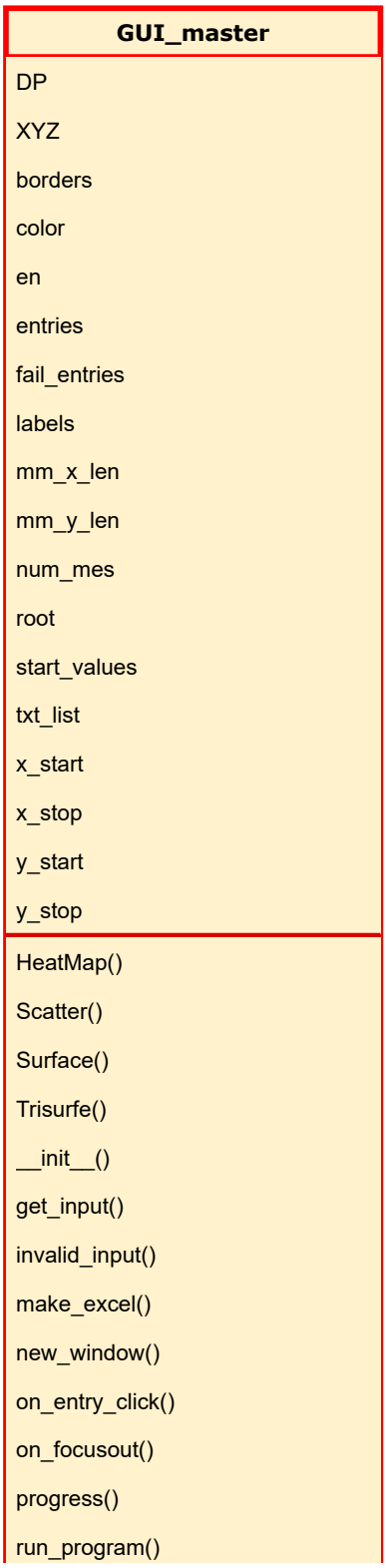


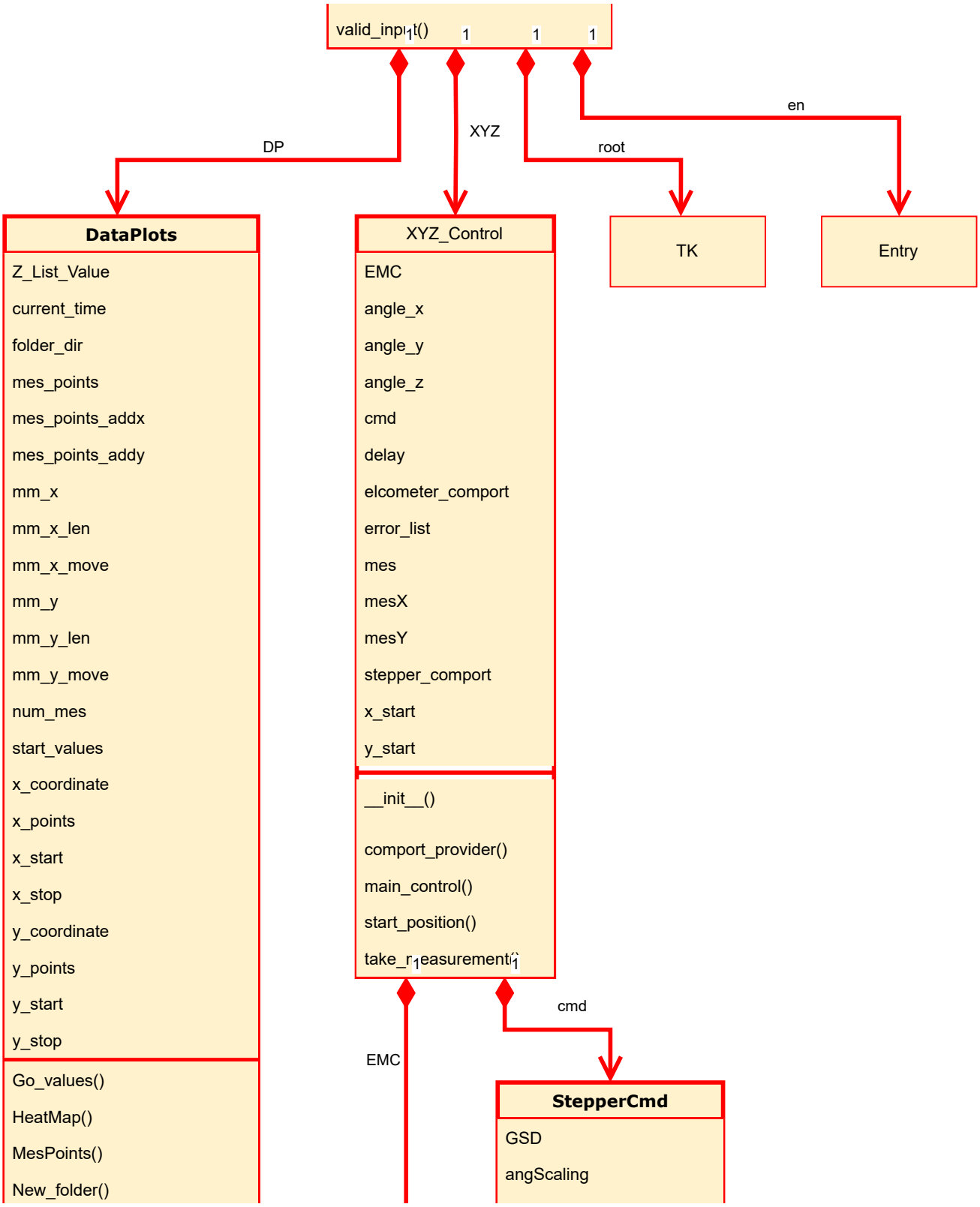
Figur C.1

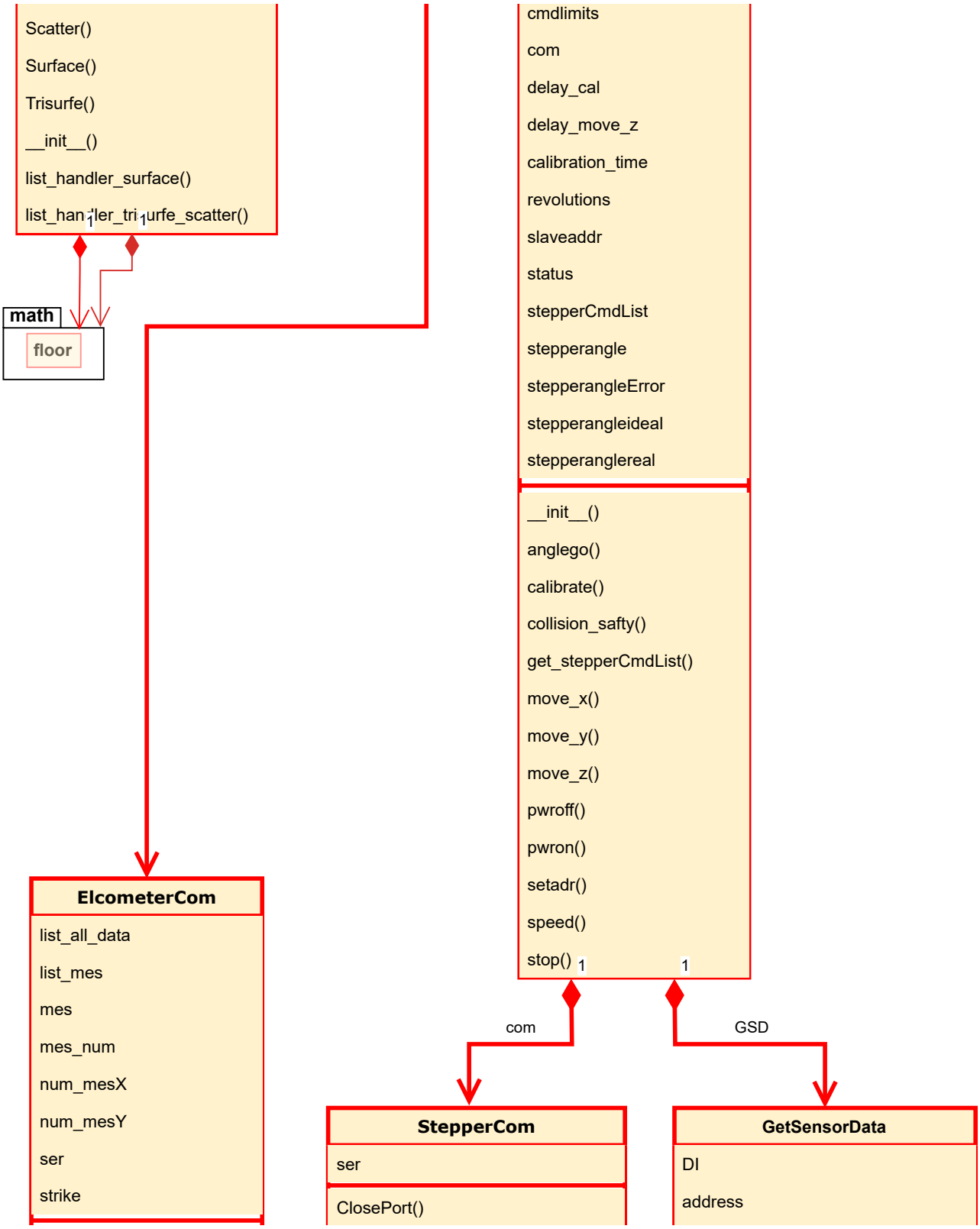


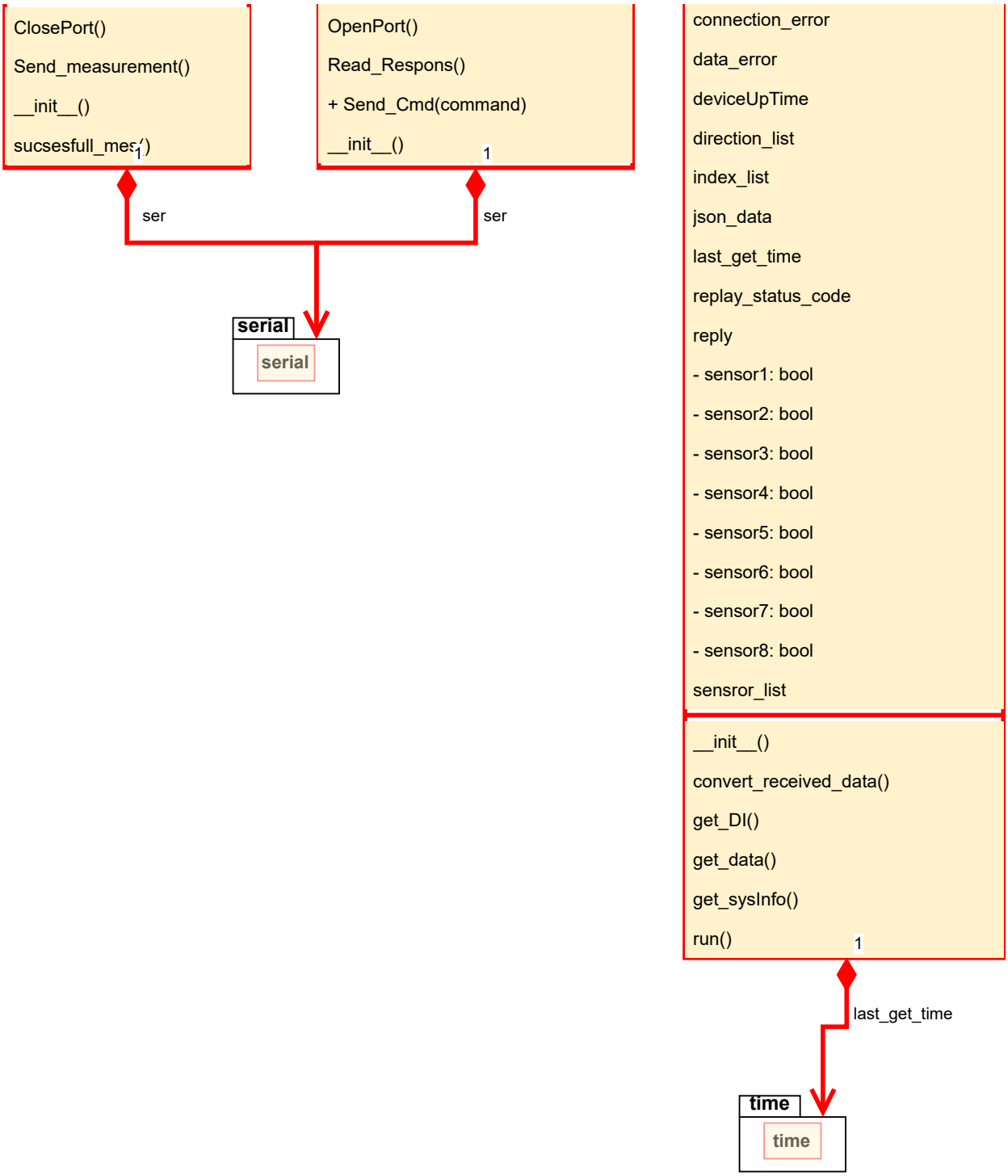
Technical parameter				
Screw diameter	16 mm	Effective length	1000 mm	
Screw lead	10 mm	Life time	10 000 km	
Accuracy	0.05 mm	Max speed (without load)	250 mm/s	
Max push force	188 N	Max horizontal speed (full load)	230 mm/s	
Horizontal load	20 kg	Max vertical speed (full load)	55 mm/s	
Vertical load	10 kg	Rate acceleration	500 mm/s^2	
Dynamic load	M_A=2.5Nm M_B=1.3Nm M_C=1.3Nm			
Motor parameter	Step angle	1.8°	Phase	2
	Rate current	2 A	Holding torque	0.95 Nm

Figur C.2: Umot dokumentasjon


GUI_master
DP
XYZ
borders
color
en
entries
fail_entries
labels
mm_x_len
mm_y_len
num_mes
root
start_values
txt_list
x_start
x_stop
y_start
y_stop
HeatMap()
Scatter()
Surface()
Trisurfe()
__init__()
get_input()
invalid_input()
make_excel()
new_window()
on_entry_click()
on_focusout()
progress()
run_program()


valid_input()
1
1
1
DP
XYZ
root
en
DataPlots
Z_List_Value
current_time
folder_dir
mes_points
mes_points_addx
mes_points_addy
mm_x
mm_x_len
mm_x_move
mm_y
mm_y_len
mm_y_move
num_mes
start_values
x_coordinate
x_points
x_start
x_stop
y_coordinate
y_points
y_start
y_stop
Go_values()
HeatMap()
MesPoints()
New_folder()
XYZ_Control
EMC
angle_x
angle_y
angle_z
cmd
delay
elcometer_comport
error_list
mes
mesX
mesY
stepper_comport
x_start
y_start
__init__()
comport_provider()
main_control()
start_position()
take_measurement()
TK
Entry
EMC
cmd
StepperCmd
GSD
angScaling


Scatter()
Surface()
Trisurfe()
__init__()
list_handler_surface()
list_handler_trisurfe_scatter()
1
1
math
floor
cmdlimits
com
delay_cal
delay_move_z
calibration_time
revolutions
slaveaddr
status
stepperCmdList
stepperangle
stepperangleError
stepperangleideal
stepperanglereal
__init__()
anglego()
calibrate()
collision_safty()
get_stepperCmdList()
move_x()
move_y()
move_z()
pwroff()
pwron()
setadr()
speed()
stop()
1
1
com
GSD
ElcometerCom
list_all_data
list_mes
mes
mes_num
num_mesX
num_mesY
ser
strike
StepperCom
ser
ClosePort()
GetSensorData
DI
address


ClosePort()
Send_measurement()
__init__()
sucsesfull_mes()
OpenPort()
Read_Respons()
+ Send_Cmd(command)
__init__()
1
1
ser
ser
serial
serial
connection_error
data_error
deviceUpTime
direction_list
index_list
json_data
last_get_time
replay_status_code
reply
- sensor1: bool
- sensor2: bool
- sensor3: bool
- sensor4: bool
- sensor5: bool
- sensor6: bool
- sensor7: bool
- sensor8: bool
sensror_list
__init__()
convert_received_data()
get_DI()
get_data()
get_sysInfo()
run()
1
last_get_time
time
time