Jan Birger Larsen

Automatisk og fleksibel posisjonering av komponenter for automatisert lodding av fleksible kretskort

Masteroppgave i Ingeniørvitenskap & IKT
Veileder: Knut Sørby
Juni 2019

Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for ingeniørvitenskap
Institutt for maskinteknikk og produksjon

NTNU
Kunnskap for en bedre verden

Jan Birger Larsen

Automatisk og fleksibel posisjonering av komponenter for automatisert lodding av fleksible kretskort

Masteroppgave i Ingeniørvitenskap & IKT
Veileder: Knut Sørby
Juni 2019

Norges teknisk-naturvitenskapelige universitet
Fakultet for ingeniørvitenskap
Institutt for maskinteknikk og produksjon

NTNU
Kunnskap for en bedre verden

Forord

Denne masteroppgaven, skrevet av stud. techn. Jan Birger Larsen, er utført vårsemesteret 2019. Oppgaven utgjør 30 av totalt 300 studiepoeng av en 5-årig sivilingeniørutdanning i Ingeniørvitenskap og IKT med spesialisering i IKT og Maskinteknikk ved Fakultet for Ingeniørvitenskap, NTNU. Jeg vil takke min veileder professor Knut Sørby for hjelp til utforming av rapporten og oppgaven. Jeg vil også takke Rune Kringstad Sandøy ved SINTEF for utforming av oppgaven, faglig veiledning, hjelp under eksperiment for delstudium og annen hjelp i form av 3d printing av nyttig komponenter for prosjektet. Til slutt vil jeg takke Erik Greiner Morset ved SINTEF for hjelp med oppsett av lysrigg og diverse hjelp i Robotlaboratoriet ved MTP Valgrinda.

Jan Birger Larsen, Trondheim, 11. juni 2019

Summary

This masters thesis document the development of a method to position flexible circuits on stiff composites with regards to many small (< 1 mm) soldering points. During this master project there was also performed a smaller study on the positioning accuracy and resolution of a UR10 industrial robot. This thesis first documents the choices of technology made for the project. Then the thesis documents the various phases of development and the different kind of methods that were tested, as well as the solution conceptually, and the implementation and physical testing of this solution. Finally the thesis documents how the problem solution was at the end of the assignment, its physical setup, its user-interface, documentation and discussion of the code used and its intended use, as well as a final evaluation of the state of the project at the end of the assignment, and some thoughts on what should be do next if it is desirable to continue developing the solution discussed in this thesis. By the end of project, testing of the positioning yielded mixed results, in some case rather accurate positioning, in other causes not. The author of this rapport is optimistic that with some changes in equipment, and some further development to remove bugs from the code, the solution could be used.

Sammendrag

Denne masteroppgaven dokumenterer utviklingen av en metodikk for å posisjonere fleksible kretskort på stive kompositter med hensyn på mange små (< 1 mm) loddepunkter. Gjennom oppgaven har det også vært foretatt et delstudium på posisjoneringsnøyaktighet og oppløsning på en UR10 industrirobot. Oppgaven går først igjennom teknologivalgene som ble gjort. Så går oppgaven igjennom de forskjellige fasene med utviklingen og hva slags metoder som ble testet, samt hva slags løsning som ble utviklet konseptuelt, og implementasjon og fysisk testingen av denne. Til slutt legger oppgaven ut for hvordan løsningen så ut ved oppgaveslutt, den fysiske oppsettingen, brukergrensesnittet, dokumentasjon og diskusjon av koden brukt og dens hensikt, samt til slutt en evaluering av prosjektets situasjon ved oppgaveslutt, og noen tanker om hva som bør gjøres videre skulle man ønske å fortsette utvikling på løsningen som er diskutert i denne oppgaven. Ved oppgaveslutt ble det gjennomført noen tester av posisjonering med miksede resultater, i noen tilfeller traff posisjonering ganske bra, i andre tilfeller ikke. Rapport-forfatter er optimistisk til at etter noen endringer i utstyr, og noe videre utvikling for å luke ut noen "bugs", kan løsningen brukes.

Innhold

1 Innledning

1.1 Bakgrunn

En norsk bedrift som blant annet produserer avanserte sensorsystemer ønsker å automatisere enkelte loddeprosesser. De aktuelle prosessene gjelder lodding av fleksible kretskort på stive kompositter, med mange små (< 1 mm) loddepunkter. Videre er det en målsetning om å gjøre dette uten bruk av produktspesifikke jigger, slik at produksjonen i størst mulig grad er tilpasset en-stykk produksjon.

1.2 Problemstilling

Masteroppgaven går ut på bruke maskinsyn for lokalisering av loddepunkter på fleksibelt kretskort og kompositt, for så å bruke en robot til å posisjonere komponentene riktig i forhold til hverandre. Selve loddeoperasjonene vil bli tatt hånd om senere i prosjektet av en lodderobot.

Oppgaven kan deles inn i følgende delmål:
• Utarbeide metodikk for deteksjon av loddepunkter på kompositt med maskinsyn
• Utarbeide metodikk for deteksjon av loddepunkter på kretskort med maskinsyn
• Utarbeide metodikk for posisjonering av komponenter med industrirobot, inkludert kalibrering
• Utarbeide metodikk for å verifisere at komponentene ligger riktig i forhold til hverandre
• Implementering og eksperimentell verifisering av metodikker i lab

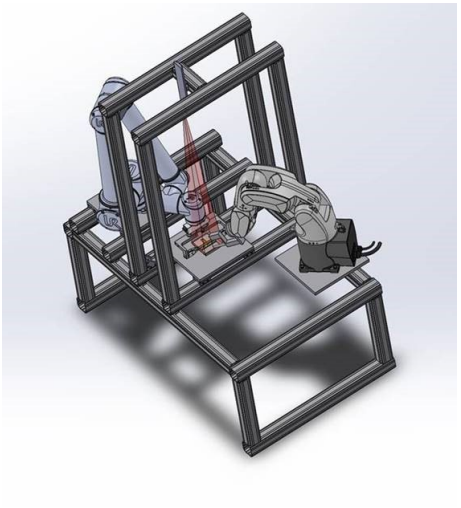
I tillegg vil det gjennomføres et delstudium for å undersøke posisjonsnøyaktigheten og minste steg-lengde for industriroboten.

1.3 Rapportens struktur

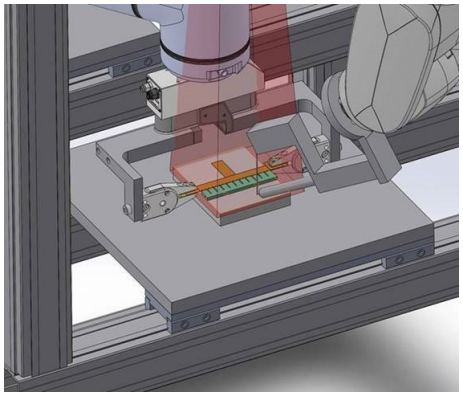
Første del av rapporten legger ut for den tidligere utviklingen i prosjektet. Etter det kommer en seksjon som legger ut for teknologien som ble brukt iløpet av denne oppgaven. Dette semesteret gjennomførte jeg også et delstudium på nøyaktigheten til en UR10 robotarm, dette delstudiumet diskuteres i seksjon 4. Videre går rapporten inn på utviklingsprosessen før prosjektets situasjon ved oppgaveslutt legges frem i seksjon 6.

2 Tidligere utvikling på prosjektet

I denne oppgaven jobbet jeg videre på et SINTEF prosjekt som allerede var litt påbegynt. Et konsept for den større løsningen (hele loddeprosessen) utenfor min del av oppgaven (posisjonering) var allerede skissert før jeg startet, og en del av valgene som tas gjennom dette prosjektet er basert på dette konseptet.



(a) Oversiktbilde over konsept for løsning, med posisjoneringsrobot og lodderobot

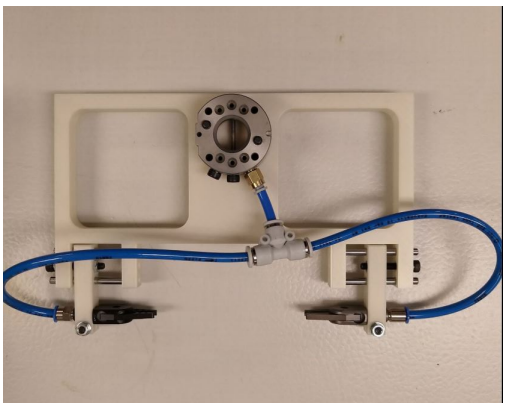


(b) Detaljbilde med lodding av en krets på en kompositt

Figur 1: Konseptskisse - 3d modell

Utenfor konseptet skissert i figur 1 ble det også diskutert løsninger som blant annet en samlebåndsorientert metode for å få til produksjon av større "batches" av komponenter av gangen, og lagringsmetoder for kretskortene, men disse tingene faller litt utenfor det som blir sett på i denne oppgaven.

For å teste implementasjonen min eksperimentelt etterhvert som jeg utviklet den, hjalp Rune Sandøy meg ved å 3d printe en enkel griper som justeres manuelt, og klemmer med pneumatisk kontroll fra robotens I/O kontroller. Det er denne griperen vi ser i figur 2.



Figur 2: Griper 3d printet av Rune Sandøy

3 Teknologibruk i prosjektet

I denne seksjonen ser jeg litt på de forskjellige teknologiene som er relevante for dette prosjektet. Jeg vil legge litt ut for spesifikasjonene, samt gjøre noen sammenligninger der andre teknologier ville vært tilgjengelige for prosjektet.

I dette prosjektet ble de fleste teknologivalgene gjort på basis av tilgjengelighet, da målet var å lage en "proof-of-concept" prototype. Den konseptuelle løsningen nevnt i seksjon 2 bruker en industrirobot og maskinsyn, og det er disse teknologiene som vil brukes i dette prosjektet. Et av målene for prosjektet er å lage en løsning som enkelt kan konfigureres for produksjon med forskjellige komponenter, så et av kriteriene for teknologivalg kan også være at det er forholdsvis enkelt å programmere inn ny produksjonskonfigurasjoner.

3.1 Maskinsyn

Maskinsyn er prosessen hvor en maskin bruker bilder til å tolke og forstå verden rundt seg. Dette er en del av kunstig intelligens feltet og en økende industri globalt. En av fordelene ved maskinsyn for produksjonsoppgaver er at det er høyt konfigurerbart og modifiserbart, så man med forholdsvis enkle kommandoer kan endre hva som detekteres i et bilde, og dermed endre måten et produkt interageres med under produksjon.

For å utføre maskinsyns-prosesser trengs noen form for kamera og maskinsyns-programvare. I dette prosjektet vil jeg bruke det som er tilgjengelig for meg gjennom NTNU og SINTEF.

3.1.1 Kamera og linse

I dette prosjektet er det viktig med et kamera med høy presisjon for å kunne presis lokalisere de små loddepunktene ¡ 1 mm. Det finnes en del forskjellige produsenter av industrielle kameraer i dag. For høy presisjon i bildene hjelper det å bruke gråtone kameraer fremfor RGB kameraer, da fargekameraer trenger 3 piksler for å representere sammen informasjon som man får fra 1 piksel i gråtone kamera. SINTEF hadde anskaffet et Basler acA4112-8gm kamera til dette prosjektet da jeg startet med oppgaven, så det var dette kameraet jeg endte opp med å bruke under utviklingen.

3.1.1.1 Basler acA4112-8gm

Basler acA4112-8gm er et Area Scan monokromatisk kamera med 4096 px x 3000 px (12.3 MP) oppløsning[1]. Ifølge spesifikasjonene skal det kunne ta bilder opp til 8 fps, som kan være relevant i applikasjoner/prosesser der det er nødvendig med hurtig bildetagning. Basler har en driver for kameraene sine kalt pylon.

Figur 3: Basler acA4112-8gm

3.1.1.2 Tamron M111FM50

Tamron lager kameralinser for maskinsyn applikasjoner. SINTEF hadde i tillegg til Basler kameraet skaffet seg en Tamron M111FM50 linse[2]. Dette er en modell i en serie laget for høy ytelse, liten forvrengning (distortion) av bildene, og ekstra god ytelse i nærfokus situasjon. Linsen som vil brukes er et kamera med spesifikasjonene 1.1 50 mm F/1.8 C, som betyr:

- 1.1 er størrelsen på bildet sendt videre til kameraet/sensoren. Basler kameraet har en sensor som kan opptil 1.1" i bilde størrelse så dette går fint.
- 50 mm er brennvidden.
- F/1.8 er blenderåpningen.
- C er hva slags type feste det er. Basler kameraet har også et c feste, så det passer fint.

3.1.2 Programvare

Det finnes flere forskjellige alternativer når det gjelder maskinsyns-programvare. Det mest utbredte er det lisensfrie OpenCV. På robotlaboratoriet er også Scorpion Vision, MVTec Merlic og MVTec Halcon tilgjengelig. Programvaren jeg vil bruke i dette prosjektet vil være MVTecs Halcon, fordi det har mest spesialfunksjoner, og er det mest velutviklede. Men Halcon har også en repeterende kostnad i form av lisens, og i den sammenheng kan det være nyttig å se om man i en evt. implementasjon faktisk vil trenge Halcon, eller om gratis program som OpenCV kan være godt nok.

Fordelene med Halcon over OpenCV:

- Mange flere funksjoner. Der OpenCV har hovedsakelig lav-nivå funksjonalitet som man med en betydelig mengde egenprogrammering og kunnskap innen

maskinsyn kan utføre avanserte maskinsyns-oppgaver, er Halcon mye mer tilgjengelig. Halcon har allerede funksjonalitet som dekker veldig mange forskjellige områder, og selv om du er uerfaren i maskinsyn er det mulig å finne fram og få til relativt brukbare løsninger.

- Halcon har et GUI som gjør det veldig lett å holde styr på de forskjellige variablene i bruk, samt et vindu som viser bildeprosesseringen i real-time, som er veldig hjelpsomt under debugging. Man kan også endre på koden i Halcon program i runtime, som er en eksepsjonelt nyttig egenskap i utviklingsprosjekter som dette.

Fordelene med OpenCV over Halcon:

- OpenCV er gratis, Halcon har lisenskostnad.
- Man kan programmere og modifisere mer i OpenCV, og dermed få bedre ytelse.
- Halcon kan også noen ganger ha bugs i kildekoden, og siden den er utilgjengelig, kan man ikke få fikset det, og må bare vente på at MVTec fikser det. I OpenCV kan man gjør slike ting selv.

Jeg vil si Halcon fungerer best for prosjekter der man ønsker at det skal være enkelt for brukere å legge til nye programmer, dvs. prosjekter med høyt krav til modifiserbarhet. OpenCV på sin side er derimot langt mer krevende å utvikle i, men dersom man skal lage en maskinsyns-applikasjon som så går umodifiserbart i all evighet, er OpenCV bedre enn Halcon, da man kan gjøre mer optimalisering. I dette prosjektet er det riktignok ikke et krav rundt hurtighet, så optimalisering er ikke så relevant. Det er også nyttig å enkelt kunne legge til kode for å prosessere eventuelle nye komponenter i framtiden uten å måtte være for erfaren med maksinsyn, så totalt sett virker det som Halcon er det bedre valget for dette prosjektet.

3.2 Robot

Bruken av industriroboter har økt kraftig globalt de siste årene[3], drevet bl.a. av overgangen til industri 4.0[4]. Fordelen er med industriroboter i den sammenheng er at de kan settes til å automatiske gjennomføre relativt kompliserte produksjonsprosesser, også kan man holde det gående lenge med rimelig stabil kvalitet på slutt resultatet. I tillegg er det mulig å programmere industriroboten til å gjennomføre forskjellige prosesser avhengig av inputen man gir, noe som gjør det veldig egnet til en-stykk produksjon.

Til dette prosjektet fikk jeg låne en UR10 robot fra NTNU/SINTEF Robotlaboratoriet ved MTP Valgrinda. Universal Robotics, som lager UR10 robotene, er verdens største selskap innen "cobots"[5], collaborative robots, roboter som kan jobbe som kan jobbe i samme fysiske arbeidsområdet som mennesker. UR10 er Universal Robotics sin største robot. For en eventuell implementasjon av denne løsningen hos sluttkunde kan det nok være nyttig med en noe mindre robot, både for mer sparsommelig areal bruk, og fordi man med mindre roboter enklere kan få høyere presisjon, men i dette prosjektet gikk det fint med en UR10.

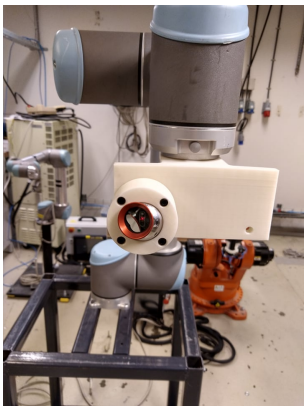
Figur 4: UR10 med kontroller

4 Delstudium - UR 10 robotarm

Som en del av prosjektet gjennomførte jeg et eksperiment, med hjelp fra Rune K. Sandøy, for å undersøke posisjoneringsnøyaktigheten og oppløsning til en UR 10 robot. Dette eksperimentet ble utformet med inspirasjon fra et eksperiment gjennomført av Young & Pickin[6].

(a) Eksperimentets oppsett

(b) Leica AT960 Medium Range lasermåler



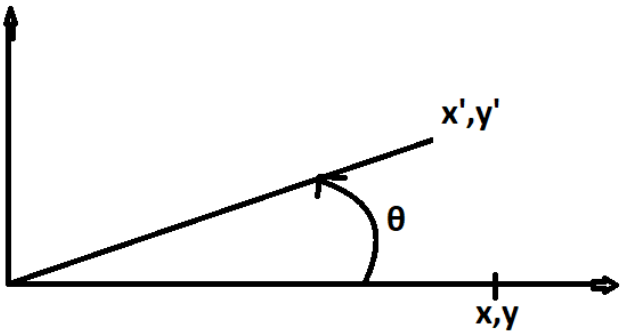
(c) Lasermåleren måler posisjonen til kula festet på roboten

Figur 5: Oppsett med UR10 robot og Leica lasermåler brukt under eksperimentet

For å måle posisjonsnøyaktigheten kjørte vi serier med lengre bevegelser (200/300mm per punkt) langs en av robotens akser. Ved vendepunktene/ytterpunktene måler vi posisjonen når roboten kommer inn fra begge retninger langs aksen. For å måle posisjonen både For å måle oppløsningen kjørte vi serier med mikrobevegelser, for å måle responsen på kommandoene. En liste over de forskjellige seriene som ble kjørt, og hvilke parametere de ble kjørt med, kan finnes i Vedlegg B.
Vi brukte en Leica AT960 Medium Range lasermåler for å finne faktisk posisjonen til robotarmen i verdenskoordinater. Disse målingene ble sammenlignet med kommandoene gitt inn, samt koordinatene målt med robotens interne koordinatmåler.

4.1 Resultater og diskusjon

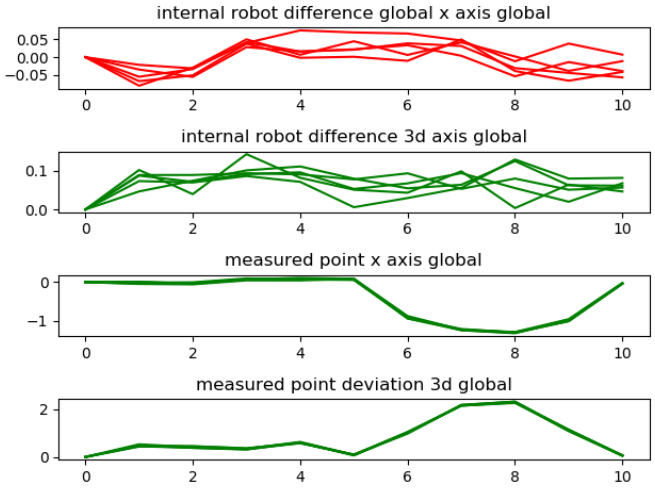
For å bruke resultatene fra Leica'ens målinger, må x og y aksene fra lasermåleren og robot kalibreres. De har samme z-akse. Transformasjonen blir derfor en enkel 2d transformasjon gitt i likning 1:



Figur 6: Forklaring av variablene i likning 1

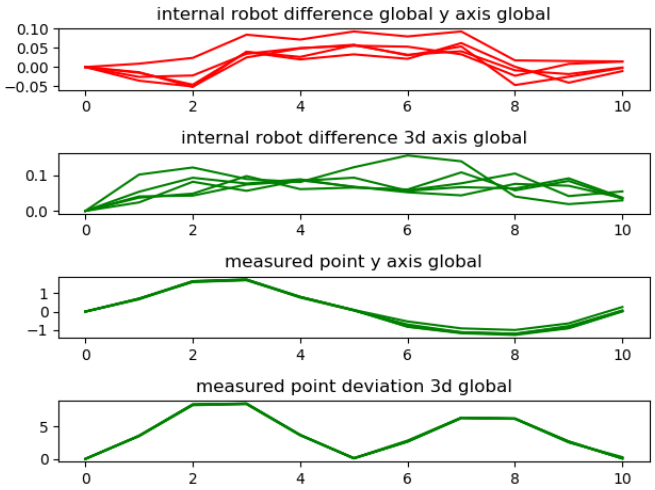
$$\begin{aligned} x' &= xcos(\theta) - ysin(\theta) \\ y' &= xsin(\theta) + ycos(\theta) \end{aligned} \tag{1}$$

4.1.1 Posisjonsavvik på større bevegelser



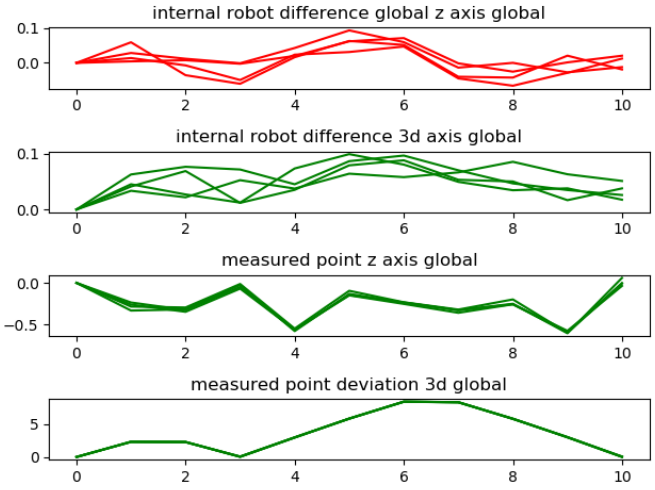
Figur 7: Grafer fra serie 1-5

I figur 7 har vi grafer fra serie 1-5. Disse målte posisjonen under bevegelse i x-akse mellom punktene 0mm, 300mm og 600mm (med vendepunktsmåling) samt symmetrisk til den andre siden. Initial posisjonen til roboten i disse seriene lå slik at roboten fint kunne nå alle posisjonene uten av å strekke seg til sine ytterpunkter. Fra den øverste grafen ser vi at posisjonsavviket langs den målte aksen ligger mellom ca -0.075 millimeter og 0.075 millimeter ifølge den interne koordinatmåleren i roboten. Den andre grafen viser posisjonsavviket i 3d fra robotens koordinatmåler. Dette avviket ser ut til å ligge på max 0.15 mm. Graf 3 og 4 viser målingene fra lasermåleren. Graf 3 gir avviket langs x aksen, som har et max på rundt 1.2 mm. Posisjonsavviket i 3d gis av graf 4 som viser max på rundt 2.1 mm.



Figur 8: Grafer fra serie 6-10

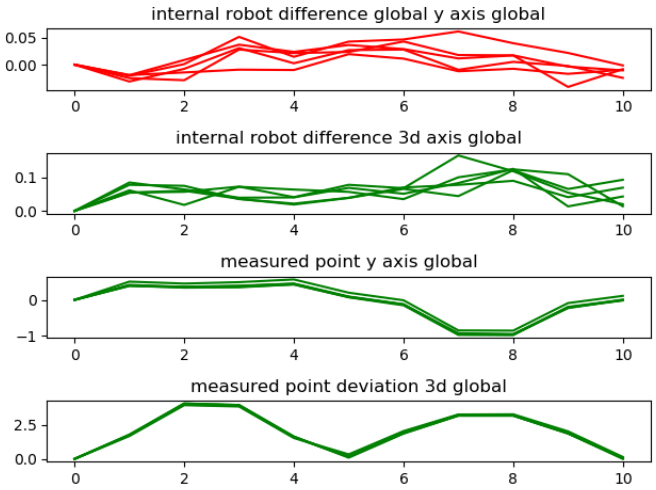
I figur 8 ser vi grafene fra serie 6-10, liknende forsøk som i serie 1-5, men for y-aksen i steden. Vi ser lignende resultater som i figur 7, med unntak av avviket funnet av lasermåleren i 3d, som blir en del høyere på nesten 8 mm. Vi ser at avviket i y aksen derimot er langt mindre, med amplituder bare noe større enn x aksen. Dette tyder på at det er større feil i x- og z-posisjon over bevegelsen, noe som muligens kommer når bevegelsen går nærmere og over basen på roboten, noe som gjør at flere ledd blir brukt. Roboten er i en mer "sammenpresset" posisjon når den går over basen. Det kan hende dette gjør at vi får mer unøyaktighet i posisjoneringen.



Figur 9: Grafer fra serie 11-15

I figur 9 ser vi grafene fra serie 11-15, som var forsøkene med globale bevegelser i z-retning. Etter som initial posisjonen brukt var den samme som i x og y, der z verdien var relativt høy, endret vi bevegelses mønsteret litt for disse forsøkene, og gikk isteden først 300 mm opp (med vendepunktsmåling) også 900 mm (0, 300mm, 600mm, 900mm) ned (med vendepunktsmåling) Avviket mellom ønsket posisjon og robotens internt målte koordinater er her ca. det samme som med x og y bevegelsene.

Det lasermålte avviket i ren z akse bevegelse har en amplitude på rundt halvparten av avvikene i x og y bevegelsene, men 3d avviket når en amplitude på nesten 8 mm her og, slik som i serie 6-10. Også her går roboten over et området med en noe "sammenpresset" positur, da den prøver å holde en fast posisjon i x-y planet, men går over en ganske lang avstand i z planet.

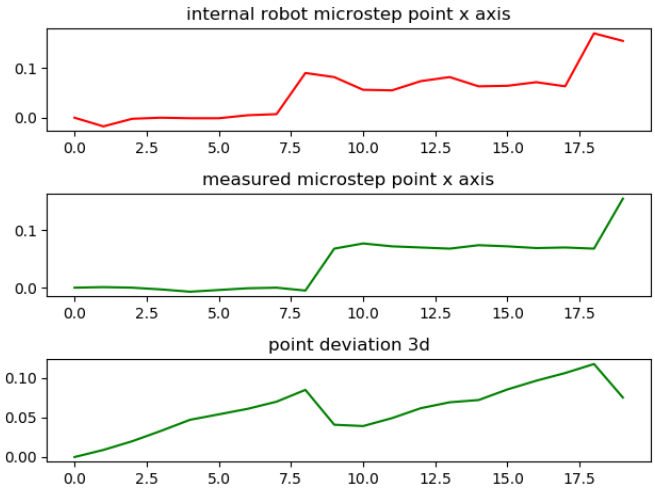


Figur 10: Grafer fra serie 32-36

De siste seriene vi gjennomførte (etter microsteppingen beskrevet i seksjon 4.1.2) var for å teste global posisjonering fra en initial posisjon som var mer utstrakt. I figur 10 ser vi resultatene etter forsøk av y bevegelse med utstrakt robot arm. Da vi først testet dette, nådde vi UR robotens "albue-singularitet". For å unngå dette under testing brukte vi derfor et bevegelses mønster med 200 mm forflyttning av gangen fremfor 300 mm. Som vi kan se er det ikke stor forskjell i nøyaktighet målt fra den interne koordinatmåleren under disse forsøkene fra de tidligere. Men noe overaskende er det at 3d avviket målt med lasermåleren faktisk er mindre enn de tidligere y-bevegelses forsøkene. Det kan muligens være fordi i denne utrakte posisjonen, så var det ikke mange større bevegelser i de forskjellige leddene over bevegelses mønsteret, mesteparten av bevegelsen skjedde i base-leddet.

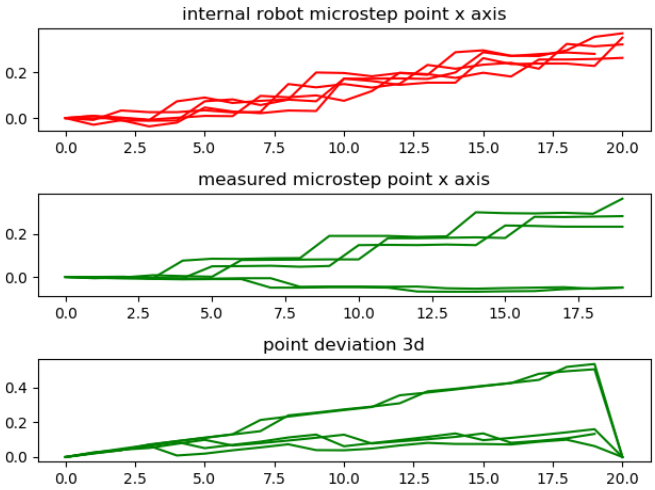
4.1.2 Steglengde på roboten

Det andre vi så på under dette eksperimentet var robotens oppløsning/minste steglengde. Dette er høyst relevant for det videre prosjektet, ettersom oppgaven er å få til posisjonering med veldig høy presisjon, og dette blir vanskelig å få til om roboten ikke klarer å bevege seg med nødvendig høy nøyaktighet. Forsøkene ble gjennomført ved å sette roboten til å gjøre veldig små inkrementelle bevegelser i en akse, for så å måle posisjonen, både med robotens interne koordinatmåler og lasermåleren.



Figur 11: Grafer fra serie 16

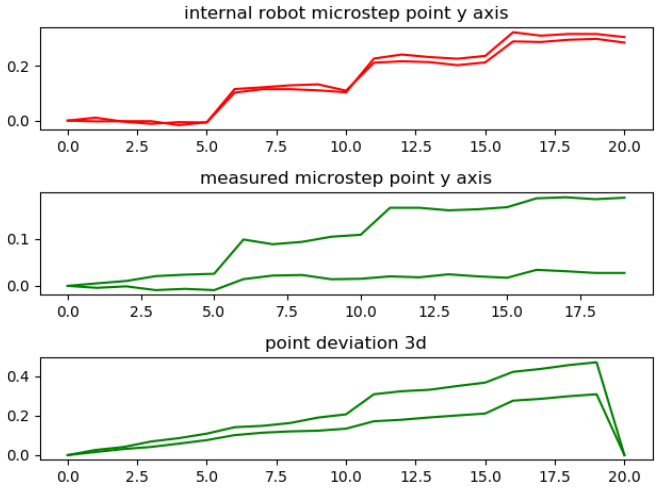
I figur 11 ser vi resultatene fra serie 16, som målte mikro steg i x-aksen. For hver måling er det gitt en økning på 0.01 mm i den ønskede x posisjonen til roboten. I den øverste grafen ser vi den internt målte x-koordinaten, og den andre grafen viser lasermålingen av x-aksen. I begge disse grafene ser vi at x-posisjonen ikke endrer seg nevneverdig før måling 8 (internmåling) eller 9 (lasermåling), da posisjonen plutselig flytter seg ca 0.08 mm. Vi ser at avviket (som er målt opp mot ønsket posisjon) øker mer og mer imens roboten står stille, men når roboten flytter seg hopper 3d avviket ned til ca 0.04 mm. Ettersom roboten etter 8 økninger på 0.01 mm beveget seg ca 0.08 mm i x-retning, tyder det på at det meste av dette resterende avviket må komme fra avvik i y- og z-posisjon. Det er også litt interessant og merke seg at det neste "fallet" i avvik, som inntreffer ved neste tilfellet av at roboten flytter seg, også er ca like stort som det første fallet. Dette kan tyde på at avviket i y- og z-posisjonen øker lineært med mikro steg bevegelsene.



Figur 12: Grafer fra serie 17-21

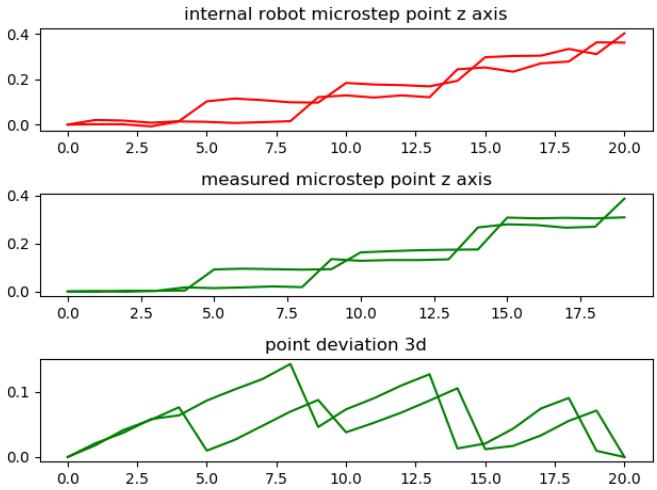
Etter serie 16 bestemte vi oss for å øke steg-størrelsen for å kunne se om vi kunne få

litt flere "hopp" i bevegelsen for å bekrefte mistankene våre. Vi gjennomførte i serie 17-21 like forsøk som i serie 16, men med en steg størrelse på 0.02 mm isteden. Vi ser i figur 12 resultatene fra disse forsøkene. Av en eller annen grunn har to av seriene gitt veldig rare resultater i lasermålingene som ikke er gjenspeilet i internmålingene. Avviket fra disse to seriene kan vi også se skyter litt i været, ettersom det virker som lasermåleren ikke har målt særlig bevegelse i x-aksen, annet enn litt bevegelse i feil retning. Men hvis vi ser bort fra disse målingene virker det som hypotesen om en steglengde på rundt 0.08 mm kan være riktig. Vi ser også her at avviket etter roboten har beveget seg virker å være veldig svakt økende, som tyder på at avviket i y- og z-koordinat er svakt økende. Fallet i avvik på slutten er falskt, da det ble lagt inn et ekstra punkt under plottingen for å unngå programmerings feil.



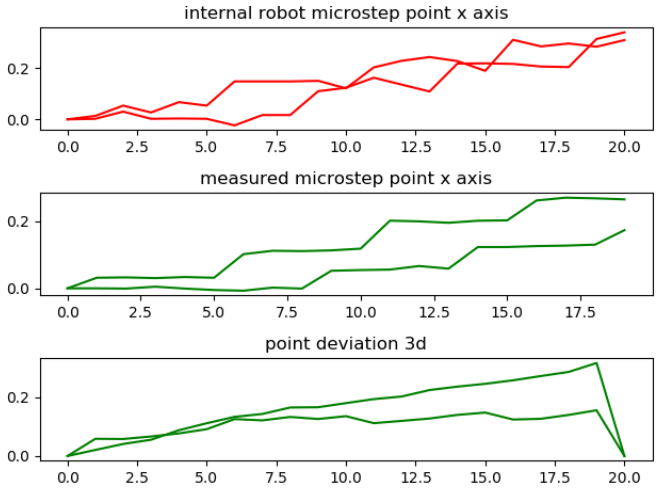
Figur 13: Grafer fra serie 22-23

Videre gjorde vi to serier med 20 steg i y-aksen på 0.02 mm, for å se om det var noen forskjell i steglengde. Resultatene er plottet i figur 13 Steglengden målt fra robotens interne koordinatsystem ligger her også på rundt 0.08 mm. Fra lasermålingene virker det å være litt større usikkerhet. Den ene lasermålingen gir ganske tydelig hopp på rundt 0.08mm, men det andre forsøket virker ikke å ha plukket opp noen bevegelse. Det mest interessante her er riktignok det at punktavviket her aldri faller, selv ikke etter lasermåleren har målt en endring i y koordinat. Dette gjør isåfall at det virker som det er merkbare avvik i x- og z- posisjonering. Et lignende problem var også tilstede under global y-akse bevegelse.



Figur 14: Grafer fra serie 24-25

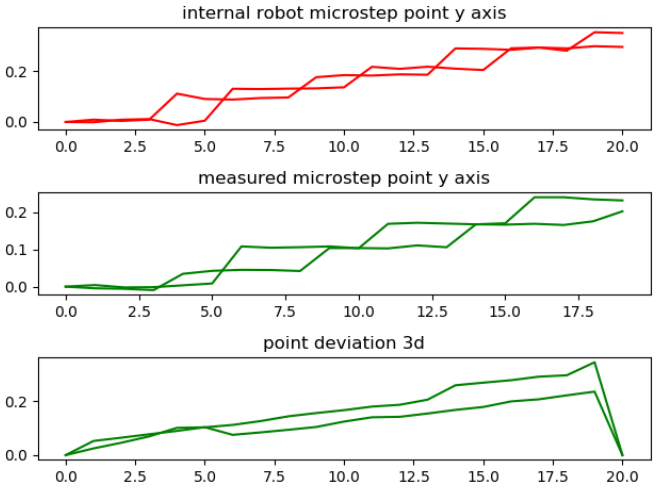
I serie 24-25 målte vi steglengden på z-akse bevegelse. Fra plottene i figur 14 ser vi at steglengden virker å være større i z retningen, rundt 0.11-0.14 mm. Internmålingene og lasermålingene kommer her mer overens enn i de tidligere forsøkene. Noe som er interessant og merke seg er at punktavviket i starten legge seg i to ganske like mønster men med forskjellige amplitude, dvs. det er større forskjell mellom de ønskede koordinatene og målte koordinatene i starten av det ene forsøket. Punktavvikene virker å nærme seg hverandre etterhvert som roboten har bevegd seg litt. Dette er et mønster som har vært tilstede i noe mindre grad i de tidligere forsøkene på x- og y-posisjon også. Dette tyder på at roboten er litt upresis med sine egne mikrosteg i starten av å gå i en bevegelses retning, men så blir mer presis etterhvert som litt bevegelse er gjennomført.



Figur 15: Grafer fra serie 26-27

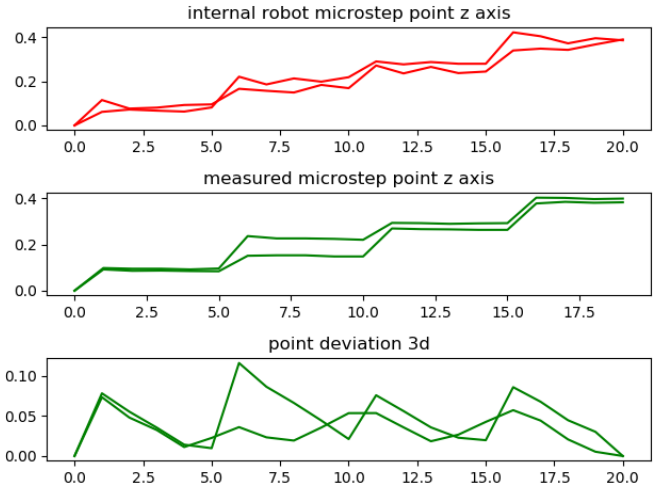
Etter å ha sett på steglengde fra en "sentral" initial posisjon, gjennomførte vi noen forsøk på steglengden hvor robotarmen var utstrakt, for å se om dette hadde noen nevneverdig effekt. I serie 26-27 kjørte vi forsøk på bevegelse i x-retningen med 0.02

mm i inkrement på ønsket posisjon. I figur 15 ser vi resultatene fra disse forsøkene. Internmålingene er en del mer ujevne i denne konfigurasjonen enn i de tidligere forsøkene. Vi ser også at det her og er det samme problemet som diskutert tidligere med at det første mikrosteget inntreffer på forskjellig inkremenet i de forskjellige seriene. Punktavviket på forsøket hvor den første bevegelsen kom ved inkrement 8 virker å øke lineært, men interessant nok holder avviket ved det andre forsøket seg ganske stabilt. Selv om punktavviket virker å ha doblet fra bevegelse fra en "sentral" initialposisjon, ser det ut som steglengden fortsatt er rundt 0.08 mm i utstrakt posisjon.



Figur 16: Grafer fra serie 28-29

Serie 28-29 gjentok forsøkene fra 26-27 i utstrakt posisjon, men denne gangen med bevegelse i y aksen, se figur 16 for resultatene. Også her er det noe "snubling" i de første mikrostegene, men så virker det å stabilisere seg rundt ca 0.08 mm, slik som i mindre utstrakt initial posisjon. Det er interessant og se hvordan punkt avviket ligger relativt nærme det det var under de første y-mikro stegene (se figur 13).



Figur 17: Grafer fra serie 30-31

Serie 30-31 var de siste mikrostegs forsøkene. Disse testet bevegelse i z-aksen med utstrakt robotarm som start-posisjon. Resultatene er plottet i figur 17 og likner ganske på resultatene i figur 14. Internmålingene er her litt mer ustabile enn tidligere, mens lasermålingene virker å tydelige indikere stegstørrelser på rundt 0.11-0.14 mm, slik som tidligere. Det er merkbart at punktavviket for de to forsøkene først ligger ganske likt/ganske ovenpå hverandre, men så stabiliserer den ene seg veldig, mens den andre ikke gjør det.

Fra disse resultatene ser det ut som den minste steglengden til UR10 roboten vi har brukt er rundt 0.08 mm i x-y planet, og rundt 0.11 mm i z planet. Det virker også som at de første mikrostegene i en ny bevegelsesretningen har større avvik enn de som kommer etter man har beveget seg litt i en spesiell akse. Alle bevegelsesretningen ble testet i etterfølgende forsøk, men det er vanskelig å si om "den første inkrements feilen" i roboten forsvant i påfølgende forsøk, selv om de gikk i samme bevegelsesretning. Mellom hvert forsøk satte vi initialposisjonen til roboten tilbake til "start" og det er mulig at denne bevegelsen evt. var bidragsytene til at feilen gjentok seg.

4.1.3 Implikasjoner for den større oppgaven

Den mest interessante informasjonen fra dette eksperimentet med hensyn på masteroppgaven, er minste-steg lengden til UR roboten, siden oppgaven handler om å posisjonere med høy presisjon. Posisjoneringen vil operere i 2d, så det viktigste er x- og y-steglengden, z posisjonsforskjeller blir stort sett for å jobbe i forskjellige plan, så det er ikke like nødvendig med høy presisjon der.

Det kan muligens vise seg problematisk at de første mikrostegene inntreffer på forskjellige kommandoer, spesielt om det trengs mikro korreksjoner for posisjoneringen. Dette er isåfall et problem som jeg tror blir vanskelig å programmere bort, da det ville bety at man trenger å kjøre større bevegelser for mindre korreksjoner, som isåfall bare erstatter første mikrostegs avvik med makrostegs avvik.

5 Utvikling

Denne seksjonen dokumenterer i hovedtrekk utviklingsprosessen for denne oppgaven. Delseksjonen er lagd opp litt etter kronologisk rekkefølge for når det ble gjort i prosjektets varighet. Noen ting ble riktignok gjort litt om hverandre, og det vil derfor ikke være rent kronologisk lagt ut for utviklingen. En del bilder i denne seksjonen vil være av konfidensielt materiale, og det vil derfor i teksten bli referert til figurer som ligger i et konfidensielt vedlegg.

Ettersom det meste av teknologien brukt i dette prosjektet var ukjent for meg gikk den første perioden i å sette seg litt inn i de forskjellige teknologiene tilgjengelig. Målet for denne oppgaven var hovedsakelig å prøve å få til en "proof-of-concept" type prototype, så mye av utviklingen gikk på å teste ut ideer for så å forkaste de eller ta de videre. Utviklingsmetodikken ble derfor hovedsakelig å utvikle på en måte som gjør at det enkelt kan "debugges" i hvert steg, spesielt siden det er flere operasjoner som samarbeider i denne prosessen. Det vil si, utviklingsmetodikken er hovedsakelig test-drevet.

For å gjennomføre denne oppgaven tenkte jeg først ut en løsning konseptuelt. Dette ledet til stegene som ble tatt i utviklingen av prosjektet. Disse stegene kan skrives som:

1. Sette opp maskinsyn-programvare med kamera, og koble videre til prosessen som kontrollerer UR roboten
2. Få kalibrert maskinsynet til å kunne finne posisjoner på bildet i verdenskoordinater
3. Lage prosedyrer for å ta og prosessere bilder av de forskjellige komponentene
4. Kode metoder som tar inn informasjon fra bildene prosessert av maskinsynprogramvaren og lager ur robot kommandoer
5. Fysisk testing

5.1 Teknologivalg

Den første delen av utviklingen utenfor det rent konseptuelle forklart i seksjonen over, var å velge ut teknologi for prosjektet. Teknologien som er valgt er litt mer utfyllende beskrevet i seksjon 3 om gjeldende teknologi.

Som nevnt i seksjon 2 var det allerede anskaffet et kamera da jeg startet oppgaven, et Basler acA4112-8gm. Til dette kameraet brukte jeg en Tamron M111FM50 linse. I tillegg fikk jeg låne en UR robot i NTNU/SINTEFS robotlaboratorium ved MTP Valgrinda. Teknologivalgene er hovedsakelig gjort avhengig av hva som var tilgjengelig i laboratoriet, i tillegg til hva SINTEF hadde tilgjengelig. Maskinsynsprogramvaren valgt var tyske MVTecs Halcon siden SINTEF hadde en lisens ledig og Halcon virket å være et veldig effektivt program.

5.2 Oppsett av kamera

En viktig del for utviklingen er at kamera oppsettet er passe høyt slik at man unngår potensiell kollisjon med robotarmen, samtidig som det er nærme nok til å få et passe detaljert bilde av komponentene, ettersom det skal jobbes på små detaljer. Etter å ha testet noen forskjellige høyder (48.7 cm, 56.5 cm, 70 cm, 78 cm) i en arbeidscelle med diffuse overlys - og dermed ideelle lysforhold, kom jeg fram til at den beste høyden for klare bilder med en grei høyde var 56.5 cm. Da jeg flyttet oppsettet over til robotcellen jeg jobbet på under resten av utviklingen, satte jeg kameraet i en høyde 51.4 cm, ettersom lysforholdene var litt dårligere i robotcellen.

5.3 Maskinsyn

Den neste delen av prosjektet gikk til å sette seg inn maskinsyn-teknikker. Jeg hadde tilgang til Halcon, og ettersom jeg ville teste dette, lette jeg etter forskjellige lokaliserings teknikker som fungerte med Halcons programmeringsspråk HDevelop. I dette prosjektet er det loddepunkter som skal lokaliseres, det vil si informasjonen vi trenger å finne er et sett med punkter. I tillegg er det slik at den ene kompositten har "loddefelter" fremfor loddepunkter, så der må det en litt annen lokalisering til å for å få det til å fungere.

5.3.1 Metoder testet for lokalisering av loddepunkt

Til punktlokaliseringen var det hovedsakelig to potensielle kandidater som utpekte seg når jeg leste meg opp på Haclon: Å bruke Halcons "Shape Matching", eller å jobbe med såkalte "regioner", dvs. alle piksler som faller innenfor en bestemt kategori.

1. Shape matching. Metode der man manuelt definerer en "form" som Halcon så leter fram i bildene man tar.

    - Fordelen med shape matching virker å være at den er veldig sensitiv til "polarity changes" - forskjell i lysstyrke - så metoden er veldig sensitiv til å finne "shapes", og derfor virker det som shape matching vil være ganske stabilt under forskjellige lysforhold. Dette er bra for å motvirke lysstøy fra evt. andre produksjonprosesser som foregår i nærheten av denne prosessen.
    - Ulempene med shape matching som jeg kom over under testingen er at det er forholdsvis mye hardkoding, med definisjon av former som skal letes etter og usikkerhet i størrelse på de. Formene slet også med å håndtere hvis man ser på former som er veldig gjentagende på en komponent. Det er noen bilder som illustrerer dette problemet i seksjonen "Arbeide med shape matching" i det konfidensielle vedlegget.

2. Thresholding med arealutvalg. Thresholding er når man oppgir noen grenseverdier for gråtoner i bilde, også separeres bilde i "regioner" hvor gråtonene er

innenfor grenseverdiene. Man jobber så litt på regionene, med bl.a. arealutvalg for å finne de riktig elementene i bilde.

- Fordelen med thresholding er at det er veldig lite hardkoding, og man kan dermed enkelt få tatt hensyn til litt feilmargin på deteksjonen i forhold til størrelse på elementer, kurving og lysforhold.
- Ulempene med thresholding s+om viste seg under testingen var at metoden er veldig sensitiv til støy, da alle piksler med en viss gråtoneverdi ble plukket opp. Samtidig var det situasjoner hvor det var piksler som skulle blitt "sett" av thresholding, som pga. skygger eller liknende endte opp med å ikke bli detektert. Dette kan medføre til at regionene som blir funnet, er "vridd" litt til den ene eller andre siden, så når man finner koordinatene for regionene - jeg bruker areal senteret til dette - så er disse koordinate skjøvet til en side. Man får derfor variasjon i koordinat posisjon iforhold til hverandre, da mengden støy ved de enkelte loddepunktene kan variere. Dette må så prosesseres senere for å vite at man faktisk har funnet de riktige koordinatene. Ettersom thresholding er veldig lyssensitivt er det også et problem med forskellige materialer i fleksprintene. Disse problemene er gitt eksempler på i seksjonen "Arbeide med regioner" i det konfidensielle vedlegget.

3. MSER - Maximally Stable Extremal Regions[7]. Dette er en spesiell form for regions deteksjon som jeg kom over litt senere i oppgaven. Poenget med MSER er at det er regioner som er laget av "homogene" området, det vil si områder i bilde med liten lokal variasjon i gråtone.

    - Fordelen med MSER er at det fungerer mye som thresholding med sikkerhetsmarginene man kan bygge inn i kodingen, men uten ulempen med høyt nivå av støy fra små lysrefleksjoner.
    - Ulempen med MSER er at det trengs en mer komplisert koding, og er sånn sett litt mindre intuitivt å jobbe med

Da jeg testet disse så jeg originalt bare på metode 1 og 2, ettersom jeg ikke kjente til MSER før senere i prosjektet. Da endte jeg med å bruke "thresholding" ettersom det var en del enklere/mer modifiserbart å jobbe med regioner enn med Shape Matching. Ulempen med regionene ble da å finne en stabil måte å fjerne mesteparten av støyet. Noe støy vil uansett prosesseres senere. Dette ble gjort via de mange "select_shape" kommandoene i Halcon, som plukker ut regioner basert på forskjellige form-karakteristikker.

Når jeg senere kom over segment_image_mser kommandoen i Halcon, fant jeg ut at denne var en langt mer effektiv metode for å lokalisere loddepunktene. Jeg fikk riktignok ikke testet den like mye ut på "loddefeltene" på kompositt 2, og gitt at hovedproblemene rundt deteksjon og lokalisering der ikke er småkornet støy og falske lokaliseringer, men heller falske lokalsering grunnet gjenskinn på forhøyningen ved loddefeltene, så er det mulig at MSER ikke er den beste løsningen for den komponenten.

5.3.2 Lysforhold og støy

Under den første testingen av kamera og maskinsynsprogramvaren jobbet jeg hovedsakelig i en celle med diffuslys ovenfra. Under disse ideelle lysforholdene fikk jeg sett litt på kapasiteten til teknologien, men når jeg skulle jobbe videre - når jeg skulle inkludere robotarmen i arbeidet - var det ikke lenger aktuelt å jobbe i denne ideelle cellen. Blant annet begrunnet med at en eventuell implementasjon hos endekunde i SINTEFS prosjekt optimalt burde trenge så lite oppsett som mulig. Så vi satte opp en robotcelle med en UR robot som jeg skulle jobbe videre på, og en montech rig for å plassere lyskildene, samt kamera.

Til lyskilder brukte jeg 2 Bar Lights som var tilgjengelig i laboratoriet. Disse var dessverre ikke av samme modell, og hadde derfor veldig forskjellig lysintensitet. Dette fungerte dårlig, og vi gikk derfor til anskaffelse for 1 til av Advanced Illuminations AL-S025300 lys, da det var denne som fungerte best på egen hånd.

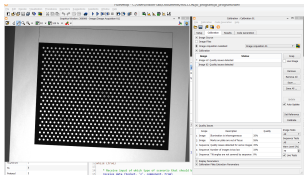
Figur 18: Advanced Illumination EuroBrite^{TM} Bar Light

Under utviklingen kom jeg over muligheten for at det kan være en fordel med forskjellig lysintensitet på bildene tatt av kompositt og de tatt av kretskortet, siden kompositten er laget av mer matt materiale enn kretskortene. Jeg rakk dessverre ikke å prøve å implementere forskjellige I/O kommandoer for lysene under de forskjellige bildetagningene.

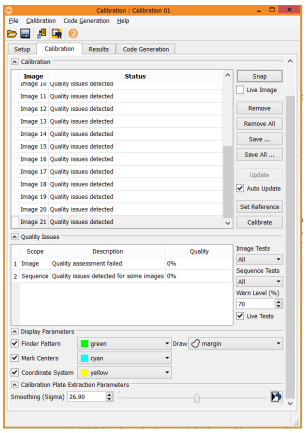
5.3.3 Kalibrering av maskinsynet

En av fordelene med Halcon, er at det har såkalt "assistants" for en del nyttig operasjoner. En av disse operasjonene er kamerakalibrering, som korrigerer for "radial distortion" (positiv distortion kalt "barrel" og negativ distortion kalt "pincushion")

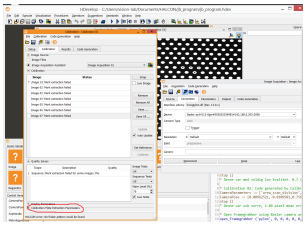
samt lager et verdenskoordinat system som gir lengde målinger istedenfor piksel målinger. For hjelp til å gjennomføre denne prosedyren har MVTec en god lærevideo på YouTube (Hvis hyperlinken ikke fungerer er url'en https://www.youtube.com/watch?v=iEjH244KRbw).



(a) Eksempel på bildet av kalibrasjons platen/arket. Grunnet ujevne lyskilder får vi ujevne bilder, som kalibrasjons assistant ikke var spesielt glad i.



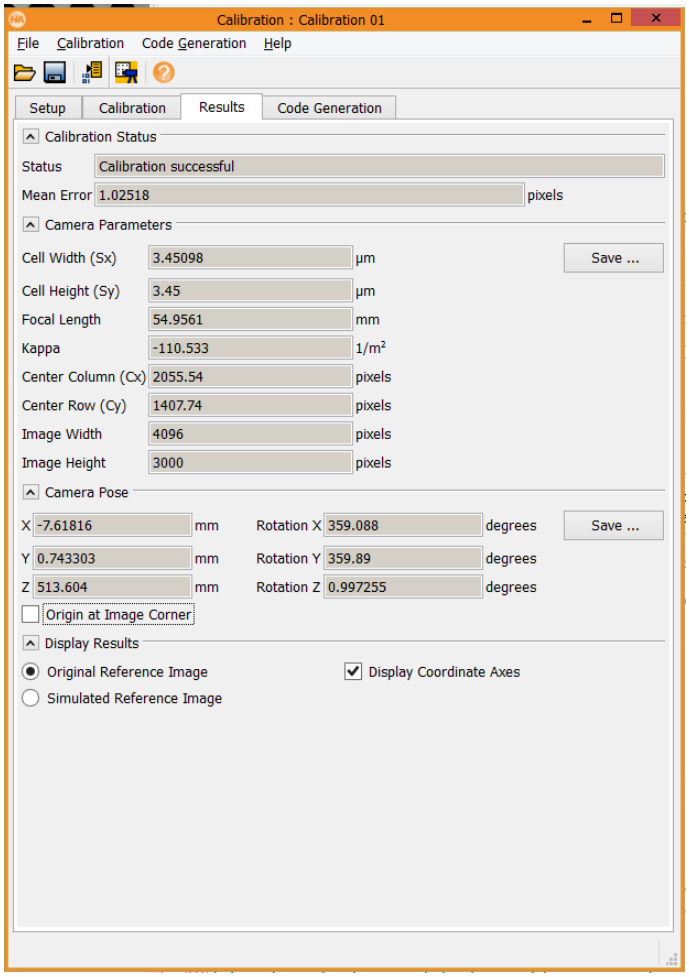
(b) Uansett hva jeg prøvde hadde alle bildene en del kvalitetsproblemer. Jeg var usikker på hvor stor innvirkning dette ville få for ende resultatet, men etter å ha prøvd en del kalibreringer virker det som det gikk greit.



(c) "Mark Extraction Failed". Dette var fordi et "smoothness" parameter, som var gjemt i en dropdown meny markert med rød sirkel, ikke var satt høyt nok.

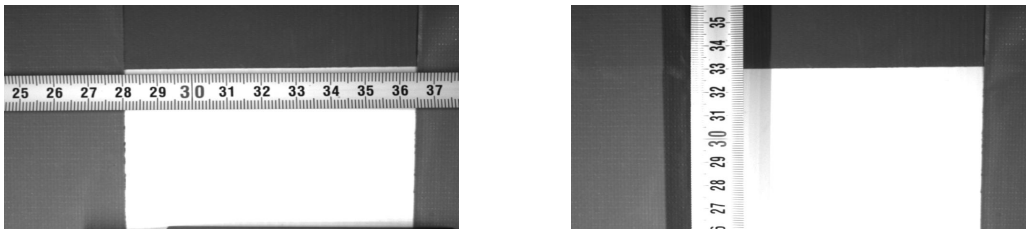
Figur 19: Diverse bilder tatt under bruk av kalibrerings assistenten.

I denne videoen var det riktignok ikke så mye informasjon rundt kvalitetsproblemer ved kalibrerings-bildene, og bildene jeg tok fikk alle status med "Quality issues detected". En stund hadde jeg også problem med "Mark Extraction Failed", men dette viste seg å være fordi "Mark Extraction Smoothness" parameteret hadde vært satt for lavt, så kalibreringen feilet på å detektere hvor skille mellom kalibreringsplatens punkter var. Dette parameteret lå litt skjult til, som vist i figur 19c.



Figur 20: Resultatet etter gjennomført kalibrering

Selv med bare rundt 30% til 40% kvalitet, hadde den endelig kalibreringen en gjennomsnittsfeil på rundt 1.025 piksler. Etter å ha gjort kalibreringen flere ganger virket det som den gjennomsnittlige feilen ofte la seg på rundt 1.02-1.07 piksler, så lenge man baserte kalibreringen på rundt 15-20 bilder.



Figur 21: Bilder med målebånd

Basler kamera er 4096 px x 3000 px. I figur 21 ser vi ca lengden (ca 13 cm) og høyden (ca 9 cm) på bilde. Dette gir en piksel størrelse på ca 0.03 mm x 0.03 mm. Dette er en grov beregning, men den viktigste observasjonen er uansett at ca 1 i gjennomsnittlig piksel feil, gir en lokaliseringsfeil på 0.03 mm, eller si 0.05 mm for litt feilmargin. Dette er en del mindre enn oppløsningen til robotarmen, så

denne feilkilden er dermed ikke spesielt relevant for oppgaven. Dessuten er nødvendig posisjonsnøyaktighet rundt ¿1 mm, så en feilkilde på ¿0.05 mm burde gå fint.

5.4 Kommunikasjon mellom prosesser

I dette prosjektet var det viktig med stabil kommunikasjon mellom prosessene for å unngå feil i informasjonen brukt for å korrigere. Det var tre prosesser som kjørte samtidig; maskinsyn ved Halcon, python for prosesskontroll og UR roboten. Jeg brukte urx biblioteket i python til å styre UR roboten, og dette biblioteket tar hånd om kommunikasjon mellom python og roboten ganske bra, selv om det noen ganger feiler på å etablere kontakt. Men da det bare trengs å etablere kontakt en gang når hele prosessen starter opp, så er ikke dette et stort problem, man kan bare prøve til det fungerer.

```
  File "C:\Users\JanBirger\venv\lib\site-packages\urx\ursecmon.py", line 344, in wait
    raise TimeoutException("Did not receive a valid data packet from robot in {}".format(timeout))
urx.ursecmon.TimeoutException: Did not receive a valid data packet from robot in 0.5
tried 11 times to find a packet in data, advertised packet size: -2, type: 3
```

Figur 22: Feil som inntreffer når urx og roboten ikke klarer å åpne kommunikasjon

Det å utvikle kommunikasjonen mellom Halcon og python tok litt mer arbeid. Valget av protokoll mellom UDP og TCP var det første å gjøre. Fordelen med UDP i denne sammenhengen ville vært at om den ene prosessen fryser for en kort periode (så lenge det ikke er under datasending) så vil dette gå helt fint, og ikke medføre krasj eller noe sånn. Men ettersom det ikke er så lett å få til god kommunisering i begge retninger med UDP, falt valget på TCP. Jeg åpnet så sockets på begge ender (Halcon og Python) med python som den lyttende. Dataen som skulle sendes fra python til Halcon var relativt greit, da dette bare var en streng som sa hva slags komponent skulle prosesseres, men fra Halcon til python ble det problematisk da det ikke var sikkert hvor mange punkter som ble funnet hver gang. For å løse dette problemet sendte jeg først over en streng som indikerte hvor mye data som skulle sendes, for så å sende dataen. Mellom hver sending fra Halcon til Python sendte jeg tilbake en "confirmed" streng, også ventet Halcon på å motta denne strengen mellom hver sending, slik at det ikke ble noen mixing av data sendt.

5.5 Prosessering av koordinater

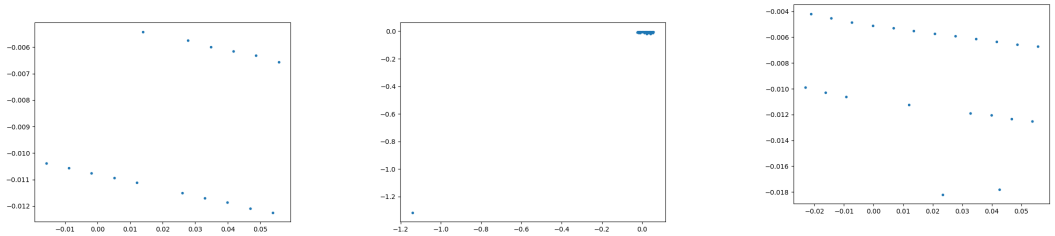
Etter å funnet koordinatene vha. Halcon var neste del av utviklingen prosessering av koordinater, for å fjerne støy og for å lage kommandoer videre til UR-roboten. Under ideelle forhold kan det være at det er mulig å programmere deteksjonen i Halcon med såpass presisjon at det vil være mindre støy enn i dette prosjektet, men for optimal robusthet i løsningen er det uansett veldig nyttig å fjerne hva enn støy det måtte være. Utviklingen av koordinat-prosesseringen fulgte hovedsakelig stegene som gås gjennom i denne seksjonen.

Et par ideer ble forsøkt. Først ønsket jeg å lage et "grid" av koordinater basert på blåkopiene av de forskjellige komponentene, hvor man så tok koordinatene fra

Halcon og koblet de opp til "gridet" for å på den måten fjerne støy og så posisjonere basert på det. Denne planen endte opp med å kreve veldig mye hardkoding og jeg endte opp med å ikke finne noen god måte å definere "start punktet" som man ville orientert resten av griddet på for å koble det sammen.

Den neste ideen jeg forsøkte var å ta flere bilder av komponentene, samt prosessere de i Halcon, for så å sammenligne de i Python for å finne områdene der Halcon hevdet loddepunktene var flest ganger, under antagelsen om at støy ville bli posisjonert spredd utover, mens de faktiske loddepunktene ville bli på plass. Denne ideen ble forkastet, da å ta mange bilder og prosessere dem ville vært veldig kostnadsfullt i form av tid. Det ville dessuten blitt en meget upresis lokalisering ellers, og mange rar edge-case problemer.

Ideen som jeg endte opp med for prosesseringen av støy, var å bruke det at loddepunktene ligger på to rette linjer. Man kan så finne punktene/støyet som avviker fra linjene. Dette vil ikke fange opp støy som ligger på linja, men det er ikke så viktig, da det som trengs for korrigering av posisjonen er vinkelen på linja og posisjon til ytterpunktene. Ideen går altså ut på å posisjonere linjer fremfor punktskyer. For komponentene som jeg kjente til holder dette greit, men dersom man i framtid ønsker å posisjonere andre typer komponenter med denne metoden, vil jeg da anbefale for eksempel å legge inn en linje for lokaliseringens skyld.



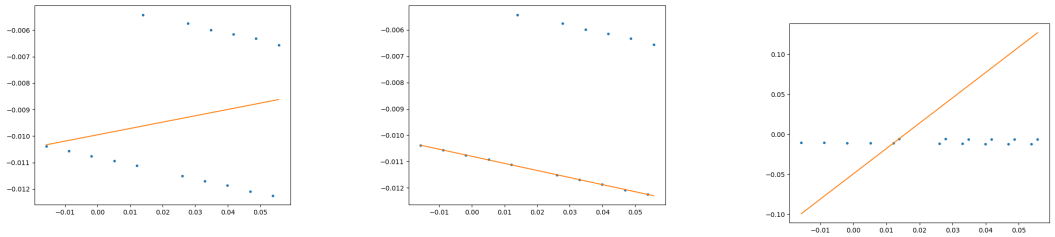
(a) Koordinater fra kompositt

(b) Koordinater fra fleksibelt kretskort

(c) Koordinater fra fleksibelt kretskort, hvor et punkt med høyt avvik er manuelt fjernet

Figur 23: Koordinater funnet vha. Halcon som ble brukt under utviklingen av prosesseringsfunksjonen. Det er støy i form av både manglende punkter og falske punkter i dataen.

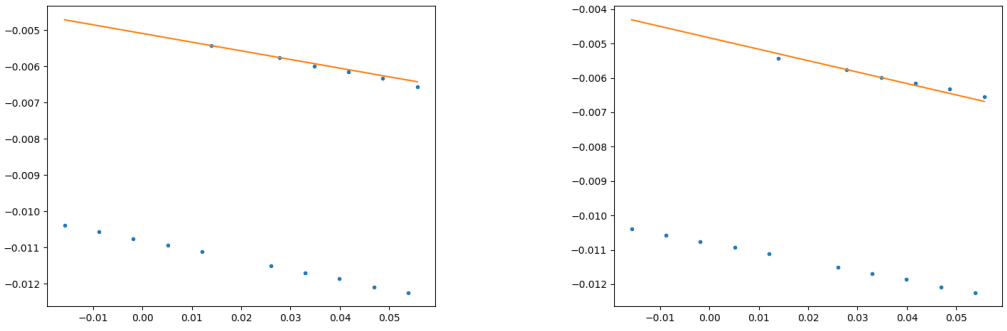
Det første som trengs for denne metoden er å finne linjene. Fra Halcon får vi inn koordinater fra prosesseringen av et bilde av en komponent. Disse punktene legger seg da f.eks slik som vist i figur 23. Ved å bruke numpy funksjonen *polyfit* over disse koordinatene, får man en linje gitt ved krysningspunktet ved y-aksen og stigningstallet.



(a) Linje interpolert fra alle punktene **(b)** Linje lagt fra de to første punktene **(c)** Eksempel på feil linje

Figur 24: I figur a) er linjen funnet av en *polyfit* over alle punktene, og i figur b) er linjen funnnet av en *polyfit* over punkt 0 og 1

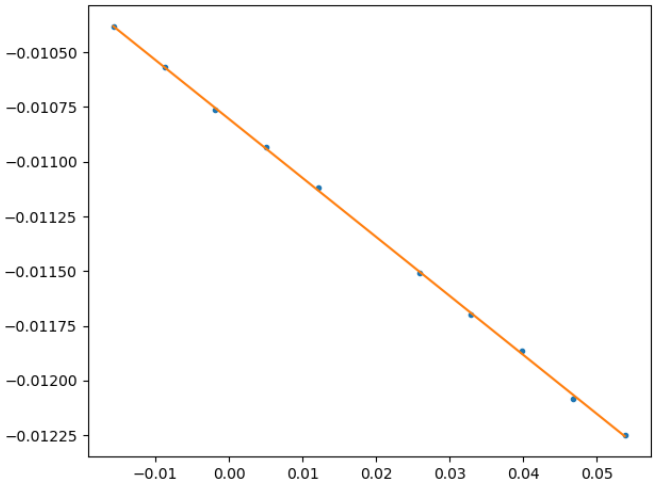
Illustrert i figur 24a ser vi problemet med å interpolere linjen fra alle punktene. 24b viser resultatet av å interpolere over bare de to første linjene. For å finne linjene må man finne de riktige punktene å lage linje fra, ellers ender man opp med et resultat som i 24c. Måten man kan gjøre dette er ved å ekstrapolere linjen fra de to punktene, også måle avstanden mellom linjen og punktene man har. Under antagelsen om at støy ikke legger seg på en linje (som riktignok kan skje, dette vil gi edge-cases man må kode for), så kan man si at dersom mer enn 3 punkter er innenfor en hvis avstand fra linja, så har man funnet enn linje med loddepunkter. Denne linja kan da lagres i form av en liste med booleans der indeksene for punktene på linja er satt til true.



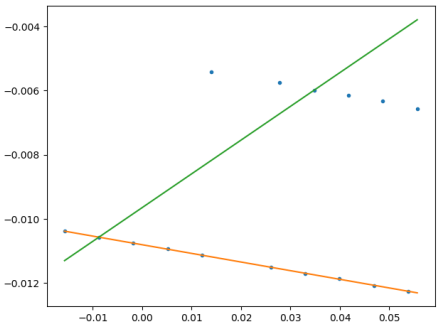
(a) Linje basert på punkt 5 og 7 **(b)** Linje basert på punkt 7 og 9

Figur 25: Eksempel på unøyaktighet på linjer som bare er basert på to punkter

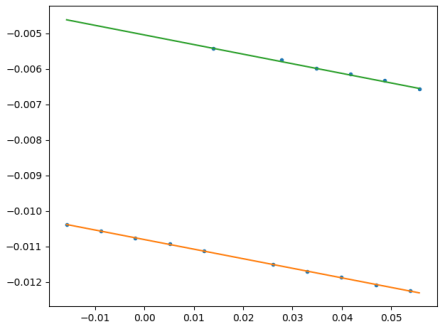
Som illustrert i figur 25 så får vi litt forskjellige linjer avhengig av hvilke to punkter vi baseres oss på. For å korrigere for denne feilen kan vi kjøre en ny *polyfit* med alle punktene i linjen når de har blitt funnet. Hvis vi sammenligner nøyaktigheten på linjen basert på 2 punkter og linjen basert på alle punktene ved å summere opp avstander får vi i dette tilfellet 0.0347 i summert opp avvik for linjen basert på to punkter og 7.967e-05 i summert opp avvik for linjen basert på alle punktene, så det er en klar fordel å gjøre denne ekstra *polyfit*en.



Figur 26: Linje interpolert fra riktig punkter i linje, de andre koordinaten fjernet for å illustrere linjens presisjon



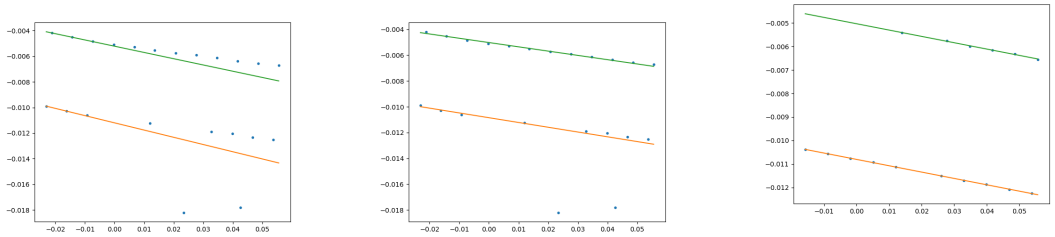
(a) Funnet 1 linje, prøver å finne den andre med interpolering fra neste punkt



(b) Funnet to linjer

Figur 27: Forsøk på å finne linje nr 2

For å finne linje nr prøvde jeg først en algoritme som gikk til neste punkt også gjennomførte samme prosessen som for å finne linje 1. Problemet jeg da kom over, illustrert i figur 27a, er når disse punktene ligger i såpass vinkel fra hverandre at punkter utenfor linjen havner nære nok linjen til å falle innenfor feilmarginen. For å unngå dette problemet, gjorde jeg sånn at når en linje er funnet, blir alle punktene på denne linjen tatt ut av vurderingen, slik at alle punkter som blir vurdert er faktiske kandidater, og man ikke kan få punkter som "sniker" seg med i beregninger. Ett mulig problem med denne løsningen er riktignok dersom det er støy som ligger akkurat langt nok unna første linjen til å ikke bli tatt ut av vurderingen, men nære nok til å sørge for at når linje mellom støypunktet og et annet punkt måles, så er det ihvertfall 2 vurderingspunkter innenfor feilmarginen av linja. Jeg tror riktignok dette ikke kan inntreffe, da jeg ikke ser hvordan man kan oppnå det matematisk. Men det kan være man burde vurdere edge-caset videre i fremtiden om man ønsker å øke presisjon.



(a) Linje interpolert fra alle punktene **(b)** Linje lagt fra de to første punktene **(c)** Økt nøyaktighet etter man korrigerer linjen basert på koordinatfiltreringen

Figur 28: I figur a) er linjen funnet av en *polyfit* over alle punktene, og i figur b) er linjen funnnet av en *polyfit* over punkt 0 og 1

Når man har funnet de to linjene fra hver sine to punkter kan man som tidligere nevnt kjøre en ny *polyfit* med alle punktene som er nære nok linja til å bli plukket opp. Deretter kan avviket minskes videre ved å gjøre enda noen iterasjoner sånn at man er sikker på at man tar alle punktene som er relevante med i *polyfit* for høyere nøyaktighet. I koden endte jeg opp med en ekstra iterasjon, da det etter dette virket å være veldig lite fortjeneste (0.08217 og 0.07074 istedenfor 0.08418 og 0.07082 i oppsummert avvik). Det kan være at det for evt. nye komponenter bør økes antall "korreksjons"-iterasjoner, men for komponentene i dette prosjektet valgte jeg 1.

5.6 Fysisk oppsett for test

Som nevnt i seksjon 5.2 og 5.3.2 ble den fysiske testingen gjennom prosjektet gjort i en robotcelle. Jeg brukte UR roboten stående på en rigg, også var kamera og lys-riggen satt i et montech oppsett som ikke var i kontakt med riggen roboten stod på for å minimalisere forstyrrelser/vibrasjoner i kamera. Oppsettet er vist i figur 33 i seksjon 6.1.

(a) 3d printet jigg for kompositt 1

(b) 3d printet jigg for kompositt 2

Figur 29: 3d printede jigger

Et av målene i dette prosjektet er at det ikke skal være satt opp produktspesifikke jigger, slik at man lettest mulig kan gjennomføre en stykk produksjon. En ide vi diskuterte (men ikke fikk implementert) under den oppgaven, var å ha alle komponentene som skulle opereres på komme inn på en én-akse maskin. På denne måten kan man enkelt legge større mengder med komponenter i kø for produksjon av gangen. Denne én-akse maskinen vil da holde de jiggene vist i figur 29, som i dette prosjektet var 3d printet. Dette er en ide som kan tas videre i framtidig utvikling om ønskelig. Når jeg gjennomførte testingen under dette prosjektet la jeg bare kompositten i jiggen, som jeg satt på bordet i et område jeg hadde markert med gaffa-teip for å få ca samme plassering hver gang jeg testet det. I en implementasjon med én-akse maskin vil man sannsynligvis kunne bestemme kompositt enda mer nøyaktig enn under denne testen, da maskinens posisjon vil kunne være ganske nøyaktig i forhold til den relativt grove posisjoneringen jeg gjorde manuelt.

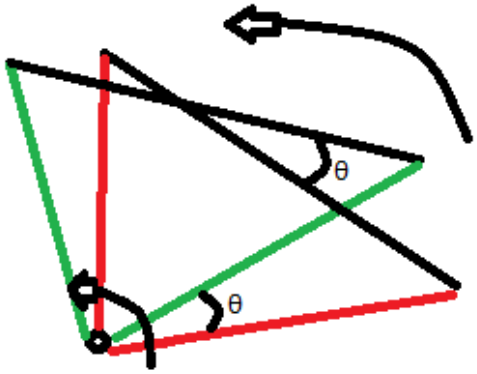
5.7 Lage kommandoer til UR roboten

Etter man har prosessert koordinatene er det neste å sende korrigeringskommandoer til UR roboten for å posisjonere bedre. For å sende kommandoer til UR robotarmen brukte jeg python med biblioteket urx, utviklet av Olivier Roulet-Dubonnet for Sintef Raufoss Manufacturing. De første kommandoene jeg bygde opp var enkle posisjons kommandoer: *initial_position()*, *standby_position()* og *placement_position()*. Ideen bak disse kommandoene er at roboten arbeider i forskjellige stadier/områder. Sånn det er kodet nå er det ingen forskjell mellom *initial_position()* og *standby_position()*, og det kan være en av de bare kan skrapes, men ideen var at roboten i *initial_position()* ville gripe en krets, mens roboten sto i *standby_position()* mens den var på standby, dvs. mellom produksjon av sett av komponenter. *Placement_position()* er første posisjonen som tas når roboten prøver å posisjonere en

krets.

For å få så presis bildeprosessering og posisjonering som mulig, jobber roboten i to plan. Med kommandoen *plane_high()* operer griperen i en høyde 5.7 mm over kompositten, mens med *plane_low()* legges kretsene i høyde med kompositten. Disse høydene er hardkodet inn og bør kanskje endres for framtidige komponenter. Før hver forespørsel til Halcon om bildeprosessering kalles *plane_low()*, mens før man skal flytte rundt på griperen kalles *plane_high()*

Ettersom man vet med ganske stor nøyaktighet hvor jiggen med kompositten i vil være posisjonert (enten av én-akse maskin eller med en annen løsning) kan den første plasseringen av kretsen på kompositten være bestemt av en hardkodet posisjon sendt til roboten. Under dette prosjektet gjennomførte jeg gripingen av fleksprintet manuelt. Avhengig av hvordan man løser gripings problem til slutt, om det blir en slags magasin løsning, eller for eksempel en samlebåndsløsning, vil gripingen av kretsen være mer eller mindre nøyaktig. I seksjon 11 i det konfidensielle vedlegget er det et eksempel av en skjev griping. Resultatet av feil i gripingen blir at posisjoneringen av loddepunktene havner i en eller annen feil vinkel i forhold til kompositten. Det første problemet er derfor å finne en måte å korrigere i feil i vinkel fra gripingen.



Figur 30: Illustrasjon over hvordan en vinkelendring i wrist 3 på roboten påvirker posisjonering av linjen med loddepunkter. Ettersom griperen fungerer som stive stenger, vris hele linjen med robotleddet. De sorte linjene er linjen av loddepunkter/kretsen, mens de røde og grønne linjene er griperen i forskjellige posisjoner. θ er vinkelendringen.

Da vi prosesserte koordinatene, fant vi linjene som gikk igjennom dem. Ettersom vi har stigningstallet på disse linjene er det en smal sak å finne forskjell i vinkel mellom krets og kompositt. Videre sammenligner jeg vinkelen på den øverste linja på kompositten med den øverste linja på krets, og den nederste linja mot nederste linja. Så ser jeg hvilket av avvikene er minst, da det er best å foreta en så liten korreksjon av gangen, i tilfelle det største avviket inneholder feil. Så sender jeg en kommando om vridning i wrist 3 for å endre vinkelen på kretsen dersom avviket er utenfor et presisjonskrav. Dette er illustrert i figur 30. For vinkel satte jeg presisjonskrav til

vinkel innenfor 0.1°. I tilfellet ved krets 1 vil dette altså bety

$$6.64 * 11 * sin(0.1°) = 0.1275mm$$

I tilfellet ved krets 2 vil dette avviket muligens være for stort, og bør derfor settes strengere krav. Men krets 2 har også flere lokaliseringspunkter, som gjør at man med større presisjon vil kunne finne posisjonene. Det er riktignok potensielt et problem med robotens oppløsning om man skal sette et enda strengere krav for nøyaktighet. Jeg prøvde med 0.5° under testingen, men da endte roboten med å hoppe mellom to punkter som begge ikke var korrekte.

Etter linjene er justert slik at de er parallelle med hverandre er den neste delen å justere så de ligger i samme høyde, at y koordinatene til punktene sammenfaller. Fra *polyfit* har vi linjenes krysningspunkt med y-aksen. Når stigningstallet (vinkelen) på linjene er de samme, så kan en høydekorreksjon være basert på forskjell i krysningspunkt. Som med vinkelkorreksjonen, sammenligner vi også her den øvre linjen fra kompositten med den øvre linjen fra kretsen, samt de nedre linjene mot hverandre. Vi tar så det minste avviket og korrigerer etter det, med en toleranse på 0.1 mm. Fra seksjon 4.1.2 vet vi at minste steglengden på roboten er rundt 0.08 mm uansett, så en høyere presisjon enn dette blir vanskelig å få til.

Når linjene er parallelle og i samme høyde, er det siste steget for en suksessfull posisjonering å posisjonere ytterpunktene i forhold til hverandre. Her er det riktignok ikke så enkelt som med vinkel og høyde, da vi ikke har noen enkelt mål for ytterpunkter, i tillegg til at selv om koordinatprosesseringen tar bort støy i høyden fra linjen, tar den ikke bort støy langs linjen. Først prøvde jeg å bare sammenligne de minste x-koordinatene fra punktene på kompositten med de minste x-koordinatene fra kretsen, for å så ta det minste avviket. Da kom jeg over problemet med at Halcon ikke alltid detekterer ytterpunktene, så man endte opp med å sammenligne f.eks. et ytterpunkt med det 3 punktet fra ytterkanten. For å unngå dette kjører jeg en løkke som går til avviket er mindre enn 6 mm, ca avstanden mellom to loddepunkter, som analyser alle punktene sånn at man kobler riktige punkter mot hverandre. Når man så har funnet side-avviket oppe og nede, tar man det minste avviket, og korrigerer for dette.

Noe som ikke har blitt tatt hensyn til under utviklingen, er om kompositten skulle flytte på seg under posisjoneringen av kretsen. Kretsen posisjoneres basert på koordinatene man finner for kompositten før krets legges på. Under testingen hendte det at griperen kom bort i kompositten, og siden den bare å la på bordet, gled den ut av posisjon. Etterhvert som man kommer til steget av utviklingen der man lager noe som gjør at kompositten holdes på plass bør riktignok ikke dette være noe problem.

5.8 Testing av løsning

Mesteparten av testingen foregikk under utviklingen, og noe av det er dokumentert i tidligere seksjoner, men ved når innleveringsfristen nærmet seg gjennomførte jeg en del tester av implementasjonen så langt den var kommet da jeg ga meg med utviklingen, og i denne seksjonen vil jeg gå litt gjennom resultatene av denne testingen.

```
  File "C:\Users\JanBirger\venv\lib\site-packages\urx\ursecmon.py", line 344, in wait
    raise TimeoutException("Did not receive a valid data packet from robot in {}".format(timeout))
urx.ursecmon.TimeoutException: Did not receive a valid data packet from robot in 0.5
tried 11 times to find a packet in data, advertised packet size: -2, type: 3
```

Figur 31: Feil når python ikke klarer å koble med UR roboten

Det første problemet som dukker opp fra tid til annen, er feil vist i figur 31. Dette er en feil som inntreffer når man forsøker å koble til roboten men feiler. Dette er riktignok ikke store problemet, det er bare å fortsette å prøve å koble til, etter et par forsøk fungerer det som regel.

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\JanBirger\AppData\Local\Programs\Python\Python37\lib\tkinter\__init__.py", line 1705, in __call__
    return self.func(*args)
  File "C:/Users/JanBirger/PycharmProjects/ur_project/gui.py", line 39, in automate_button
    main.automatization_manager(self.components)
  File "C:/Users/JanBirger/PycharmProjects/ur_project\main.py", line 371, in automatization_manager
    composite_processed = coordinate_processing(coord_composite,composite)
  File "C:/Users/JanBirger/PycharmProjects/ur_project\main.py", line 231, in coordinate_processing
    x_2 = np.array(list(compress(x,line_2_fil)))
UnboundLocalError: local variable 'line_2_fil' referenced before assignment
Grip circuit manually, then press enter to continue
[[-0.004652956166506056, -0.010833678778257812, array([0.01332343, 0.05508592, 0.0410918 , 0.03419902, 0.00636609,
       0.02723238, 0.04812121, 0.04108538]), array([-0.00477008, -0.00506311, -0.00544048, -0.0048635 , -0.00471947,
       -0.004814  , -0.00500549, -0.00543475])], [-0.010405709072696044, -0.007638903036744999, array([ 0.02556388, -0.01622766, -0.00232931,  0.01161582,  0.03248169,
        0.0464128 ,  0.00464709,  0.03945609, -0.00927432,  0.05338147]), array([-0.01058517, -0.01029442, -0.01039437, -0.01050678, -0.01063905,
       -0.01076315, -0.01043394, -0.01070206, -0.01032523, -0.01083167])]]
```

Figur 32: Ustabilitet i koordinatsprosesseringen

Når man så tester løsningen her fungerer mye greit, men noen "bugs" blir litt gjengangere. I figur 32 ser vi en "bug" som inntreffer noen ganger, der koordinat prosesseringa feiler, ved at den ikke klarer å finne linje nr. 2. Men merkelig nok, i neste forsøk, uten noen endringer, så går det helt greit. Dette er nok noe man bør se nærmere på ved videre utvikling for å øke stabiliteten på prosedyren.

Når det kom til testingen av den faktiske posisjoneringen var det også et par "bugs" som dukket opp. "Bug" nr 1 var at noen ganger når kretskoordinatene ble prosessert, så ga dette en enorm forskjell i vinkel mellom kompositt og krets, som gjorde at roboten prøvde å korrigere med å svinge altfor langt. Jeg satte inn en ekstra sjekk av vinkel før man korrigerte, slik at hvis programmet mente at den skulle korrigere med en vinkel $\geq 5.7°$, så gå den inn en rotasjon på $0.5°$ isteden, for så å analysere kretsen på nytt og forhåpentligvis ikke gjøre samme feil igjen.

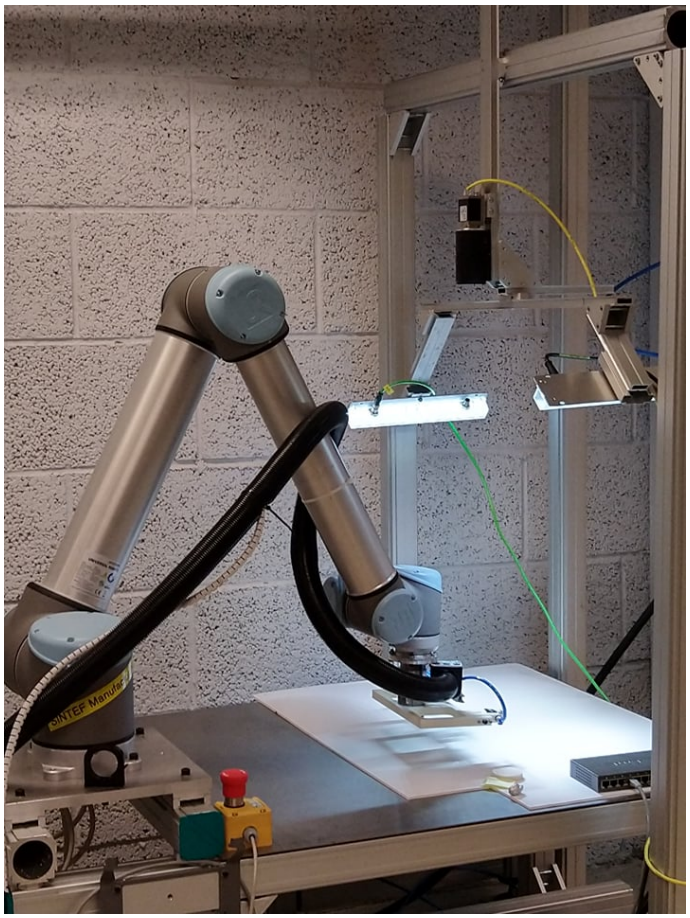
"Bug" nr 2 var også relatert til vinkelkorreksjonen. I noen testforsøk så klarte ikke roboten og plassere krets tilfredstillende nære, så den bare fortsatte å vri fram og tilbake med en veldig liten rotasjon. Dette problemet dukket også litt opp for bevegelse/posisjonering av x- og y-koordinatene også. Jeg tror dette kan være et problem med robotens minste-steg, diskutert i seksjon 4.1.2 og 4.1.3. Jeg tviler på at dette problemet er koblet til Halcon, da denne skal ha en nøyaktighet på rundt 0.03 mm. Det kan være riktignok være at denne feilen er et resultat av begge feilkildene samtidig, da python er satt til å godta posisjoneringen innenfor 0.1 mm, men hvis man både treffer maksimalt uheldig fra robotens steg størrelse på rundt 0.08 mm og Halcons feil på 0.03 mm samtidig, kan det være at dette akkurat går utenfor pythons toleranse hver gang.

Det siste problemet som virket å dukke opp var potensiellt en akkumulering av feilkilder, da posisjonering godtok en posisjonering som bommet litt i noen av punktene. Flere feilkilder kan ha bidratt til dette, blant annet at kretskortet ikke ble holdt nok i spenn fra griperen, at oppløsningen på UR roboten ikke var nøyaktig nok. Det er også mulig at man fikk maksimal krasj med Halcons presisjon, dette kan komme fra at det er større unøyaktigheten i sidene på bildene, der forvrengningen er kraftigst. Det var gjerne i disse områdene man også endte med upresis posisjonering.

6 Situasjonen ved oppgaveslutt

I denne seksjonen vil jeg legge ut for hvordan den nåværende løsningen ser ut og hvor prosjektet ligger an etter min oppfatning. Seksjonen er skrevet litt som en bruksanvisning for hvordan utvikle videre, i tillegg til at det i seksjon 6.2 er en bruksanvisning for hvordan programmet kjøres nå.

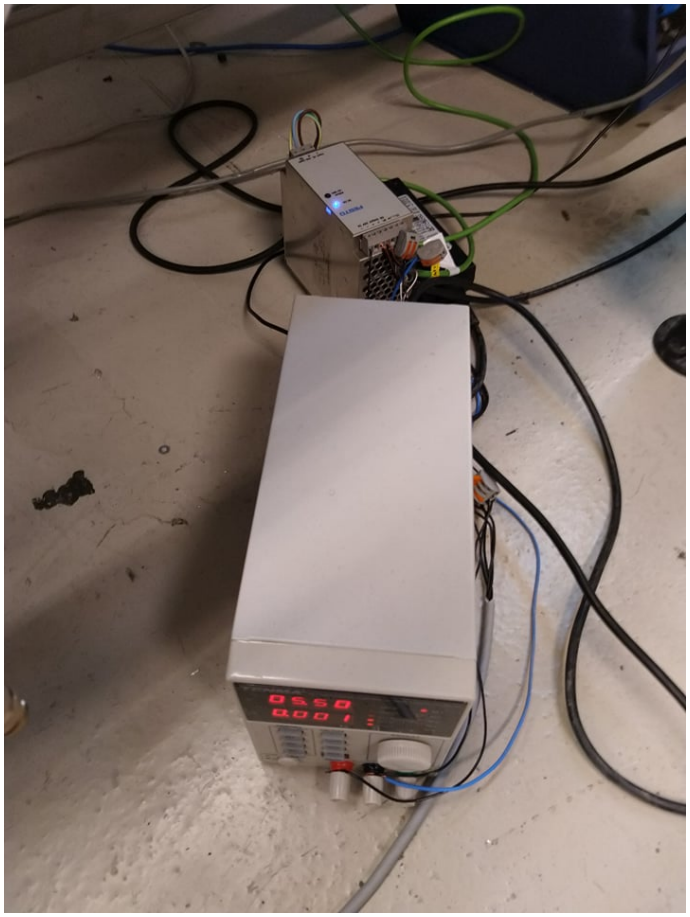
6.1 Fysisk oppsett



Figur 33: Oppsett med kamera, lysrigg og robotarm

Det fysiske oppsettet brukt for testing under prosjektet var som vist i figur 33. Griperen designet av Rune Sandøy var satt i verktøyholderen til UR roboten og hadde pneumatisk kobling som gjorde så den kunne gripe og holde kretser. Kamera og lysrigg var festet på et montech oppsett som hadde egne bein, sånn at vibrasjoner fra robotens bevegelser ikke vil gi spesielt med forstyrrelser for bildetagningen.

Lysene er vinklet innover mot komponentene for å gi optimalt med lys, men med så matt refleksjon som mulig. Når jeg prøvde å sette lysene rett ved siden av kamera skinnende rett ned på komponentene ble refleksjonen fra reflekterende overflater for kraftig til å jobbe med.



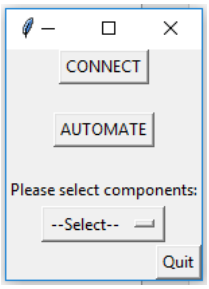
Figur 34: Dimmer og strømkilde for lysriggen

Lysene er satt opp med strømkilde og en enkel dimmer for å teste forskjellig lysintensitet. Som nevnt i seksjon 5.3.2 virket det som det kunne være nyttig å bruke forskjellig intensitet under forskjellige bildetagninger, men når jeg testet hadde jeg stort sett dimmeren bare stilt på 5.5 V. Det er denne intensiteten det nåværende Halcon programmet er kodet for.

6.2 Bruksanvisning

GUIen slik den er nå fungerer slik: Når man starter programmet (kjører gui.py), klikker man på "Connect", og den prøver å koble til UR og Halcon. Når den har koblet til UR (som noen ganger feiler, da må man bare prøve igjen), så kjører den/står den stille til man har kjørt Halcon programmet, som da kobler seg på socketen åpnet fra python. Når denne prosessen er fullført, velger man hva slags "komponentkombinasjon" man vil ha fra drop-down menyen hvor det står "Please select components". Foreløpig er UR kommandoene bare kodet inn for "Circuit1+Composite1", det andre valget "Circuit2+Composite2" har bare maskinsyn prosesseringen. Etter man har valgt komponenter, klikker man "Automate" som starter prosessen. Da kjører UR først opp i en "standby" posisjon, der man kan manuelt sette inn riktig krets i griperen, ettersom denne prosessen ikke er automatisert enda. Python programmet ber om å trykke "Enter" i terminalen for å fortsette, og resten av po-

sisjoneringen skjer av seg selv.



Figur 35: Gui brukt under testingen

6.3 GUI

Et veldig enkelt gui er satt opp for å kunne initialisere enkelt tester av lokalisering og posisjonering. Den endelig løsningen skal være automatisert, og en bedre GUI da vil være en som tar inn f.eks. en sekvens av symboler som indikerer hva slags komponenter som skal orienteres videre, hvordan "batchet" er som skal produseres er lagt opp.

Gui'et bruker python bibilioteket tkinter. Koden setter enkelt opp en tkinter Application, som inneholder en Frame der alle Widgets, i form av knapper og nedrullsmenyer, er festet. Når man klikker disse knappene fyrer Widgeten sin "command" del. "Commandene" settes da til de forskjellige relevante operasjonene - "CONNECT" forteller main klassen at den skal kjøre metoden open_and_connect_socket() (som er nærmere forklart i seksjon 6.6.2), mens "AUTOMATE" kaller metoden "automatization_manager(component)" i main klassen. Parameter "component" som sendes som input i "automatization_manager(component)" er bestemt av valget tatt i nedrullsmenyen.

6.4 Kommunikasjon mellom prosesser

For å åpne kommunikasjon mellom Halcon og python prosessen åpnes det en socket i python programmet som så bindes til ip-addressen på pcen som kjører Halcon. Python socketen står da åpen og lytter etter en socket som ønsker å koble seg på. Med Halcons open_socket_connect kommando åpnes det en socket som kobler til python. Jeg bruker TCP4 protocol på socketen, da den skal holdes åpen/kommunisere kontinuerlig. Halcons prosess opererer i en løkke der programmet står på en "receive_data" kommando mens python prosesserer data og opererer UR. Hver gang python trenger ny data, sendes det en kommando til Halcon inneholdende hva slags komponent som skal prosesseres/analyseres. Denne bildetagningen og prosesseringen blir da gjennomført, før dataen skal sendes tilbake til python prosessen. For å forsikre at all dataen blir sendt, og at den blir lagret på riktig måte, sendes det først en streng med 'd#' der # er antall elementer/bytes som skal sendes over.

6.5 Maskinsyn - programvare

For bildeprosesseringen i dette prosjektet bruker jeg Halcons egne språk HDDevelop. Koden ligger i filen "picture_processing.hdev" og den går som følger:

1. Først leser den inn kamera parameter funnet vha. Halcons kalibrering assistent. Disse parameterene må bestemmes på nytt hvis man lager et nytt oppsett. Hvordan jeg gjennomførte kalibreringen er beskrevet i seksjon 5.3.3. Disse parameteren er for å bestemme verdens-koordinater samt og motvirke forvrenging av bilde.

2. Så åpner den en "framegrabber", et objekt som håndterer bildetagningen. Denne framegrabberen styrer Basler kameraet vha. pylon driveren.

3. Det neste som gjøres er å definere en metrologi modell. Denne modellen brukes til å finne verdenskoordinatene til områdene som blir detektert. Den bruker kalibrerings parameterene til å lage et plan uten forvrenging som kan brukes til å bestemme verdens-kooordinater.

4. De siste som gjøres under "setup-fasen" er å åpne kommunikasjon med python. Sånn koden er skrevet nå er addressen som open_socket_connect på ip-adressen som jeg brukte under prosjektet, denne må åpenbart skrives om til hvilken ip-adresse man måtte bruke selv. Socketen kjører nå TCP4 protokoll, dvs. TCP med IPv4 addressering, så om pcen som kjører Halcon og python er den samme, så kan man bare bruke loopback-addressen '127.0.0.1'. Python må også lete på denne adressen isåfall.

5. Prosess løkken i programmet starter med en kommando receive_data(..) som venter på å motta data gjennom socketen åpnet tidligere. Denne kommandoen fungerer som en standby, da den ikke sender programmet videre før den mottar data. Dataen som blir mottatt blir lagret i component variabelen, som skal være en streng som forteller hva slags type bilde som skal prosesseres. Bildet blir så prosessert, forskjellige typer prosessering er diskutert i seksjon 5.3.1. Sånn koden er nå er prosesseringen av kompositt 1 og krets 1 de eneste som gir verdens-koordinater. For å få tak i verdens koordinater for krets 2 kan man nok bare kopiere koden for krets 1 og endre på noen parameteret for å tilpasse seg lysforholdene rundt krets 2. Kompositt 2 er litt vanskeligere grunnet det at man ikke kan detektere loddepunkter, men loddefelter. Hvis man på en pålitelig måten kan splitte ut de riktige regionene kan man bruke area_centre operatøren for å få regionenes areal senter i pikselplassering, som man så kan konvertere til verdenskoordinater vha. metrologimodellen som brukt for de andre komponentene. Dette vil riktignok kreve noe videre utvikling for å finne ut av.

6. Når man har funnet sett med koordinater, sendes denne dataen til python programmet. Dette skjer ved at man først sender over en streng med informasjon om hvor mange koordinater som er funnet, slik at python vet hvor mye data den skal motta, og så, etter man har mottatt bekreftelse på at strengen

kom fram til python, sender man koordinatene i form av rad-nummere (x-koordinater) og kolonne-nummere (y-koordinater), med bekreftelsesmelding fra python mellom hver data batch.

7. Den siste delen av koden er operatører som lukker framegrabberen og socketen, samt sletter metrologi modellen. Ideen er riktignok at Halcon programmet kommer til å være kjørende "for alltid", da målet for dette prosjektet er automatisering, og i mellom produksjon av komponenter så vil Halcon stå i standby og vente på neste kommando. Det finnes en "exit-clause", dvs. når Halcon mottar kommandoen "exit" går den ut av løkka og gjennomfører disse tre kommandoene, men sånn python koden er nå blir aldri "exit" sendt.

6.6 Python program

Python programmet fungerer som et mellomledd mellom kamera og roboten. Koden håndterer oppkoblingen mellom alle prosessene, robotens forskjellige arbeidsposisjoner, kommandoer til roboten, koordinat prosessering og håndtering av kommando rekkefølge for den automatiserte prosessen. Koden er delt opp i forskjellige funksjonalitetsdomener, og i denne seksjonen vil jeg gå litt gjennom hva som gjøres i hvert domene. Koden kan finnes i filen "main.py" i mappen ur_project i vedlagt zip-fil.

6.6.1 GLOBAL VARIABLES

De globale variablene håndterer *socket*en mellom python og halcon, tilkoblingen til Halcon *conn*, portnummeret til socketen *addr*, og tilkoblingen til UR roboten *rob*.

6.6.2 COMMUNICATION BETWEEN PROCESSES

Metodene i denne seksjonen håndterer operasjoner på hovedsakelig socket koblingen opp mot Halcon. Videre er en kort beskrivelse av hver metode:

- open_and_connect_socket(): Denne metoden åpner en socket og kobler opp mot Halcon. I tillegg kobler den variabelen *rob* til UR roboten vha. urx-biblioteket. Nå er adressen socketen er koblet til satt for oppkoblingen mellom dataen som jeg brukte på laboratoriet, men hvis begge prosessene (Halcon og python) kjører på en pc bør man heller bruke loopback adressen. Robotens oppkobling er også satt til adressen på roboten jeg brukte på laboratoriet.

- close_socket(): Denne metoden lukker socketen, og fjerner koblingen til variabelene *conn* og *addr*. Denne metoden kalles riktignok aldri, da det ikke er noen lukkemulighet for programmet enda.

- send_component_data(component): Denne metoden tar inn component, en streng, som input, den sender så videre denne strengen til Halcon. Etter man har kjørt denne metoden bør man kjøre receive_coordinates umiddelbart etterpå, for å motta dataene fra Halcon. Man må riktignok ikke gjøre dette, om man heller vil kjøre andre operasjoner imellom går dette fint, da Halcon holder på dataen til python åpner for å motta den, men man kan isåfall ikke kalle Halcon til å gjøre andre ting før dataen er lest/mottatt.

- receive_coordinates(): Denne metoden følger etter en send_component data for å lese ut resultatet fra Halcon. Hvis denne ikke kjøres vil halcon bare stå å vente på sende dataen. Metoden returnerer en liste med punkter i form av tupler.

6.6.3 ROBOT POSITIONS

Denne delen av koden håndterer posisjoner for roboten under setup og standby. Disse posisjonene er hardkodede, og må kodes spesifikt for oppsettet man bruker. Dette er relativt enkelt, da man bare kan manuelt posisjonere roboten slik man ønsker i i de forskjellige posisjonene, også lagre denne posisjonen. Koden inneholder også høyde nivået for de to arbeidsplanene, "høyt" og "lavt".

- initial_position(): Posisjonen der jeg tenker at den automatiserte gripingen av krets skal foregå. Når jeg testet implementasjonen under utviklingen ble gripingen foretatt manuelt.

- standby_position(): Denne metoden skal kalles når roboten går i standby. Akkurat nå er den kodet med sammme posisjon som initial_position, men i en faktisk ende-implementasjon bør nok denne posisjonen være forskjellige fra initial_position for å unngå kollisjon med løsning for automatisk griping av krets.

- initial_placement_position(): Denne metoden plasserer kretsen i posisjonerings området. Metoden gir flere etterfølgende kommandoer til UR roboten, siden banen har flere mellom posisjoner for at kretsen kommer inn på kompositten på riktig måte. Den første posisjonering er i "posisjoneringsplanet", slik at man med en gang kan ta et første bildet for å korrigere for gripefeil.

- plane_high(): Denne kommandoen flytter roboten slik at z verdien i posisjonen til roboten er -0.23775. Dette er den hardkodete høyden for "arbeidsplanet" i denne prosessen, planet der posisjoneringen skjer. Z-verdien på dette planet bør skrives om hvis man bruker en annen fysisk oppstilling enn den brukt her.

- plane_low(): Denne kommandoen flytter roboten slik at z verdien i posisjonen til roboten er -0.2434, 5.65 mm lavere enn "arbeidsplanet". Dette er "posisjoneringsplanet", der man tar bilder av kretsen for å finne ut hvilke korrigeringer som er nødvendig.

6.6.4 ROBOT COMMANDS

I denne delen av koden er det noen metoder som gir spesifikke type kommandoer til roboten. Disse metodene krever mindre input enn urx kommandoene, og får gjennomført de nødvendige opppgavene for dette prosjektet.

- move_gripper(x, y, angle): Denne metoden tar inn en x og y forflytning, samt enn vridning *angle*. X og y translasjonen gir en forflytning av verktøysenterpunktet, som medfører en lik forflytning av alle kretsens koordinater. Vinkelen

angle gir hvor mye rotasjon man vil ha i "wrist 3" på roboten, noe som roterer griperen om verktøysenter-punktet. Å gi inn en kommando med ren vinkel vil dermed også medføre en bevegelse i x og y koordinater, men disse blir korrigert for senere i posisjonerings prosessen.

- close_gripper() og open_gripper(): Kommandoer for å skru pneumatikken for griperne av og på. Ettersom gripingen i det gjeldende oppsettet må gjøres manuelt, brukte jeg ikke disse kommandoene spesielt mye, da jeg heller brukte kontrollskjermen til UR en for dette.

6.6.5 COORDINATE PROCESSING

Denne delen av koden omhandler koordinatsprosesseringen, samt avgjør hvilke kommandoer som skal sendes til UR roboten for å posisjonere kretsen bedre. Funksjonaliteten er splittet i to metoder:

- coordinate_processing(coordinates, component): Denne metoden tar inn et sett med koordinater, samt en streng som sier hva slags komponent som skal prosesseres. Foreløpig har det ikke blitt lagt inn noen komponentspesifikke prosesseringsteknikker i koden, men i framtiden kan denne strengen brukes til å avgjøre hva slags teknikker som skal brukes. Metoden prosesserer så koordinatene etter prosedyren som er konseptuelt lagt fram i seksjon 5.5. Metoden returnerer så en liste, som inneholder to lister, en for hver linje, og i hver av disse listene er: Krysningspunkt med y-aksen for linja, stigningstall for linja, x-koordinater for punktene på linja og y-koordinater for punktene på linja.

- align_composite_circuit(composite_processed, circuit_processed): Denne metoden tar inn den prosesserte dataen for kompositten, som er skaffet i starten av posisjoneringsprosedyren, og den prosesserte dataen fra det nyeste bildet av kretsen. Metoden bestemmer så hvilke kommandoer som skal sende videre til UR roboten. En konseptuell beskrivelse av disse valgene kan finnes i seksjon 5.7.

6.6.6 AUTOMATED PROCESS

I denne delen av koden håndteres den automatiserte posisjoneringsprosessen. Metoden automatization_manager(component) blir kalt fra gui'et for å gjennomføre en posisjonering. Metoden tar inn en streng, component, som sier hva slags komponenter som skal posisjoneres. Den sender en kommando til å roboten om å innta initial posisjonen, hvor den blir stående til man manuelt har satt riktig krets i griperen og trykket på Enter. Den ber så om et bilde av kompositten og sender dataen den får fra Halcon til prosessering. Den prosesserte dataen fra kompositten blir lagret i *composite_processed*. Deretter sendes en kommando til UR roboten om å innta posisjoneringsplanet. En kommando om å ta og prosessere et bilde av kretsen blir sendt til Halcon, og resultat dataene blir motatt i variabelen *coord_circuit*. Koordinatene blir så prosessert, plane_high kommandoene blir kjørt, for å flytte griperen til "arbeidsplanet", og de prosesserte dataene blir sendt til align_composite_circuit(composite_processed, circuit_processed) for å avgjøre hvilken

kommandoer som skal kjøres. Hvis kommandoer trengs å kjøres returner denne metoden verdien true, og vi går inn i en løkke med prosessering av bilde med krets samt korrigering av posisjonering, fram til align_composite_circuit(composite_processed, circuit_processed) ikke finner flere kommandoer å kjøre. Det er også et maks tak på 7 iterasjoner med korrigering før den gir opp, da det ved så mange forsøkte korreksjoner er det noe som har gått galt. Tilslutt sendes det en kommando om en siste bildeprosessering av kretsen, for å se om koordinatene nå er riktige. Det mangler en verifiseringsmetode som sammenligner punkt koordinatene, og bestemmer hva som skal gjøres ved eventuell suksessful eller mislykket posisjonering.

6.7 Sluttvurdering av løsningen

Etter min vurdering er ikke denne løsningen presis nok enda til å gjennomføre de høypresisjons posisjoneringen som den skal. Jeg mener riktignok fortsatt at det skal være mulig å få det presis nok med denne løsningen, da Halcons deteksjon virker å gi veldig presise resultater (med en feilmargin på rundt 0.03 mm). Det virker for meg som de største feilkildene kommer ifra roboten og ujevnheter i lysforholdene. Disse krever isåfall en hvis endring i fysisk oppsett, samt at om man bruker en robot som ikke kan styres med urx-biblioteket, må man omskrive delen av python programmet som omhandler robot kommandoene. Det er også mulig at de feilmarginene jeg satte opp for koordinatprosesseringen og lignende burde vært finjustert videre, men jeg er usikker på om dette vil ha noen stor effekt, da de fleste marginene ligger ganske nære minste steg-størrelse for URen allerede.

6.8 Veien videre

Dersom det er ønskelig å utvikle videre på denne løsningen, vil jeg anbefale som første steg og prøve å feste kompositt delen fast til noe, i tilfelle dette korrigerer nok for små feil til at posisjoneringen blir mer presis. Det neste kan være å forsøke å bedre lysforholdene, slik at punktdeteksjonen blir jevnere, som gir roboten bedre koordinater å jobbe etter. Man kan også prøve å involvere en robot med noe høyere oppløsning enn URen for å se om det minsker noen av presisjonsproblemene.

I tillegg er det i python programmet en serie med "TODO"s, som dekker flere av edge-casene jeg ikke fikk kommet til i løpet av denne oppgaven. Det viktigste som trengs å programmeres etter min oppfatning er en bedre verifiseringsmetode. Problemet med metoden slik den er nå er at den bare ser på kretsens koordinater, og sammenligner de med kompositten originale koordinater. Det er mulig man kan fjerne feilkilden her ved å forsikre at kompositten ikke kan flytte seg, men verifiseringsmessig er det uansett fordelaktig om man kan på en eller annen måte få detektert og sammenlignet begge settene med koordinater på en gang.

Senere bør det bestemmes hva slags kommandoer som skal tas som resultat av verifiseringen av posisjoneringen. Jeg ser for meg at en vellykket verifisering vil ledde til at roboten slipper taket på kretsen og så trekker seg unna, så komponenten kan tas videre, og en feilslått verifisering sender programmet tilbake til posisjoneringsløkken

for å prøve mer. Det kan riktignok være at man bør kode inn en "fail-safe" for tilfeller der roboten absolutt ikke klarer å posisjonere kretsen. Ellers er det noen bugs, blant annet diskutert i seksjon 5.8, som man kanskje kan få fikset. Foreløpig er koden bare velfungerende når man ser på komponenter med loddepunkter (krets 1, krets 2 og kompositt 1), men ikke for komponenter som har "loddefelter" (kompositt 2). Dette kan man utvikle for videre. Jeg vil isåfall anbefale å prøve om man kan detekter posisjoneringer for loddepunktene til krets 2 på en meningsfull måte i Halcon, slik at koordinatene som sendes til python kan tolkes på samme måte som for komponent kombinasjon 1.

Referanser

[1] *acA4112-8gm - Basler ace*. Basler AG. URL: https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace/aca4112-8gm/.

[2] *Mega-Pixel Machine Vision Lens Series*. Tamron. URL: http://www.tamron-usa.com/IO/IndOp/prod/assets/pdfs/M111FMxx_2015.pdf.

[3] *Executive Summary World Robotics 2017 Industrial Robots*. International Federation of Robotics. URL: https://ifr.org/downloads/press/Executive_Summary_WR_2017_Industrial_Robots.pdf.

[4] Frauke Muth. *Hva er egentlig industri 4.0? Innovasjonsbloggen*. Innovasjon Norge. URL: https://innovasjonsbloggen.com/2015/10/22/hva-er-egentlig-industri-4-0/.

[5] Ash Sharma. *Universal Robots Just Sold Their 25,000th Collaborative Robot*. Universal Robots Continues to Dominate Cobot Market but Faces Many Challengers. URL: https://www.interactanalysis.com/universal-robots/.

[6] Ken Young og Craig G. Pickin. «Accuracy assessment of the modern industrial robot». I: *Industrial Robot: An International Journal* 27 (6 2000), s. 427–436. URL: https://doi.org/10.1108/01439910010378851.

[7] *segment_image_mser (Operator)*. MVTec Software GmbH. URL: https://www.mvtec.com/doc/halcon/13/en/segment_image_mser.html.

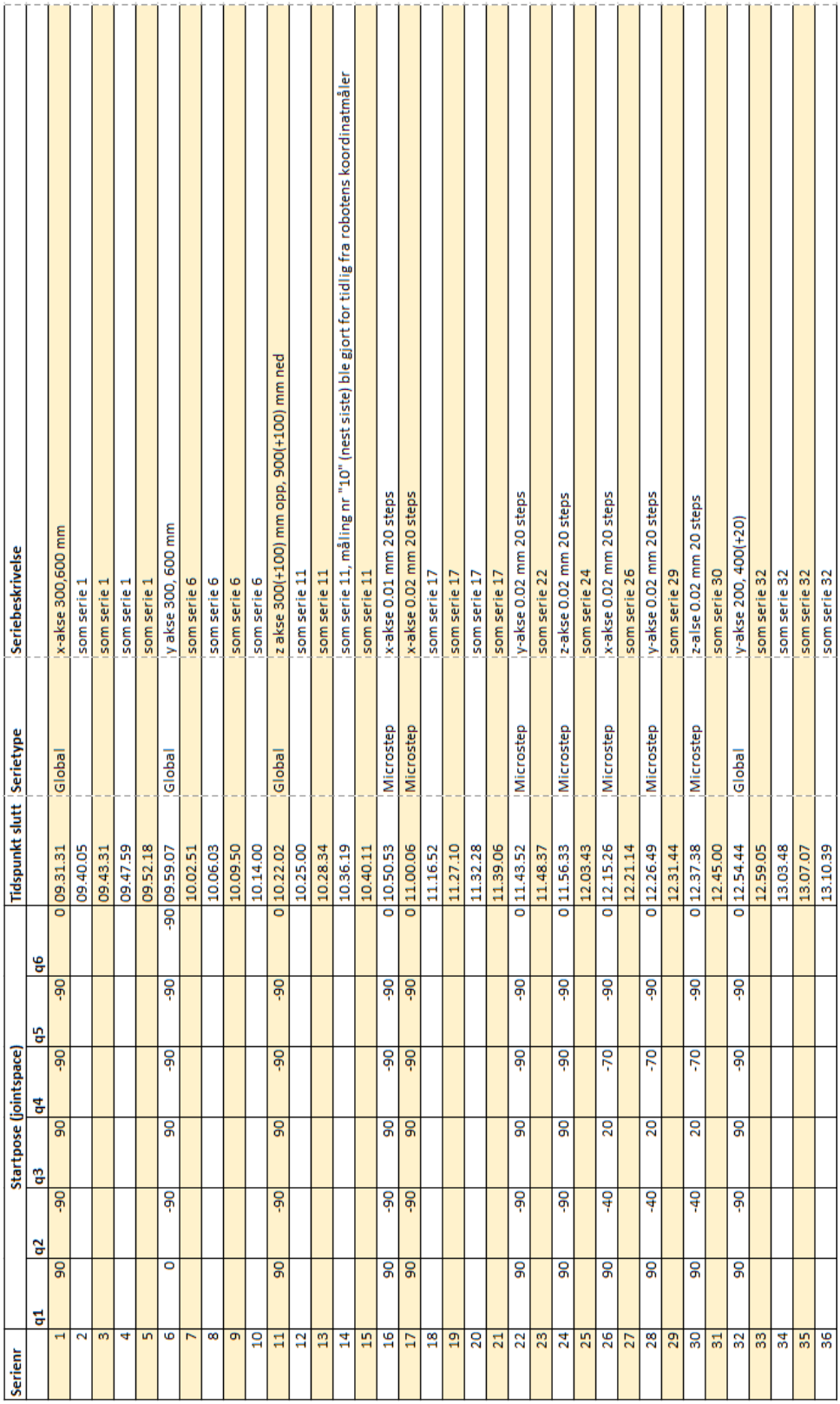
Vedlegg A

Dette vedlegget lister opp hvilke filer som hører med denne oppgaven, og kort om hva de inneholder:

- En zip fil kalt "Jan Birger Master Thesis - attachment". Denne inneholder 3 mapper:
  - Analysis: inneholder .csv og .xlsx filene fra delstudiumet, samt koden brukt til å analysere disse.
  - jb_program: inneholder Halcon koden i .hdev format, samt en ren tekstfil med samme koden
  - ur_project: dette prosjektet inneholder python koden utviklet for styring av ur og prosessering av koordinater i dette prosjektet.
- Et konfidensielt vedlegg med bilder fra utviklingsprossess.

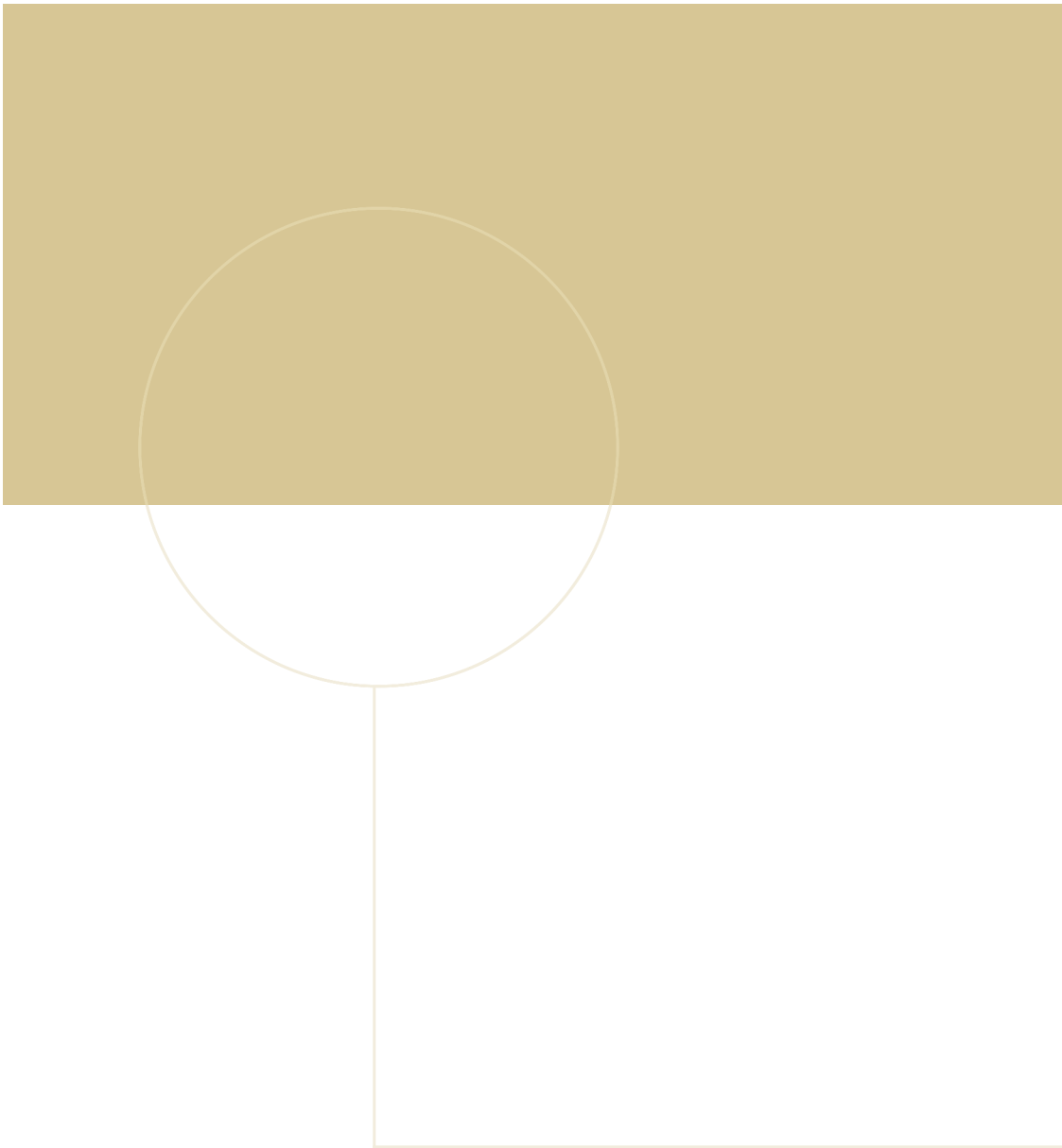
Vedlegg B

I dette vedlegget er det en liste over de forskjellige seriene fra eksperimentet i seksjon 4, med initial posisjoner, og hva slags type serie det er. Denne listen finnes også i en Excel fil kalt "UR-nøyaktighet-Datasamlingsdokument" i mappen Analysis i zip filen "Jan Birger Master Thesis - attachment".



Serienr	Startpose (jointspace)						Tidspunkt slutt	Serietype	Seriebeskrivelse
	q1	q2	q3	q4	q5	q6			
1	90	-90	90	-90	-90	0	09.31.31	Global	x-akse 300,600 mm
2							09.40.05		som serie 1
3							09.43.31		som serie 1
4							09.47.59		som serie 1
5							09.52.18		som serie 1
6	0	-90	90	-90	-90	-90	09.59.07	Global	y akse 300, 600 mm
7							10.02.51		som serie 6
8							10.06.03		som serie 6
9							10.09.50		som serie 6
10							10.14.00		som serie 6
11	90	-90	90	-90	-90	0	10.22.02	Global	z akse 300(+100) mm opp, 900(+100) mm ned
12							10.25.00		som serie 11
13							10.28.34		som serie 11
14							10.36.19		som serie 11, måling nr "10" (nest siste) ble gjort for tidlig fra robotens koordinatmåler
15							10.40.11		som serie 11
16	90	-90	90	-90	-90	0	10.50.53	Microstep	x-akse 0.01 mm 20 steps
17	90	-90	90	-90	-90	0	11.00.06	Microstep	x-akse 0.02 mm 20 steps
18							11.16.52		som serie 17
19							11.27.10		som serie 17
20							11.32.28		som serie 17
21							11.39.06		som serie 17
22	90	-90	90	-90	-90	0	11.43.52	Microstep	y-akse 0.02 mm 20 steps
23							11.48.37		som serie 22
24	90	-90	90	-90	-90	0	11.56.33	Microstep	z-akse 0.02 mm 20 steps
25							12.03.43		som serie 24
26	90	-40	20	-70	-90	0	12.15.26	Microstep	x-akse 0.02 mm 20 steps
27							12.21.14		som serie 26
28	90	-40	20	-70	-90	0	12.26.49	Microstep	y-akse 0.02 mm 20 steps
29							12.31.44		som serie 29
30	90	-40	20	-70	-90	0	12.37.38	Microstep	z-alse 0.02 mm 20 steps
31							12.45.00		som serie 30
32	90	-90	90	-90	-90	0	12.54.44	Global	y-akse 200, 400(+20)
33							12.59.05		som serie 32
34							13.03.48		som serie 32
35							13.07.07		som serie 32
36							13.10.39		som serie 32

Figur 36: Seriene kjørt under eksperimentet beskrevet i seksjon 4



NTNU

Kunnskap for en bedre verden