Høgskulen på Vestlandet

BACHELOROPPGAVE:

BO24EB-02 PLS LiDAR

Ola Sørvik Hernes
Malin Ødegård Gustavson
Viktor Søderholm

19. mai. 2024

Dokumentkontroll

Rapportens tittel: BO24EB-02 PLS-LiDAR	*Dato/Versjon* 19. mai. 2024/1.0
	Rapportnummer: B024EB-02
Forfatter(e): Ola Sørvik Hernes Malin Ødegård Gustavson Viktor Søderholm	*Studieretning:* AUT21
	Antall sider m/vedlegg 52
Høgskolens veileder: Tom Kjøde	*Gradering:* Åpen
Eventuelle Merknader: Vi tillater at oppgaven kan publiseres.	

Oppdragsgiver: Controlteam AS	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson(er) (inklusiv kontaktinformasjon): *Christian Nordbø* *48994865* *cno@controlteam.no*	

Revisjon	**Dato**	**Status**	**Utført av**
0.11	12.04.24	Første utkast	Viktor Søderholm
0.12	30.04.24	Andre utkast	Ola Sørvik Hernes, Malin Ødegård Gustavson, Viktor Søderholm
0.13	14.05.24	Tredje utkast	VS, OH, MG
1.0	19.05.24	Endelig rapport	VS, OH, MG

Ola Sørvik Hernes

Malin Ødegård Gustavson

Viktor Søderholm

Forord

Dette er en rapport for vår avsluttende bacheloroppgave våren 2024 ved Høgskulen på Vestlandet (HVL). Oppgaven er beskrevet som et system der en sensor i samspill med en PLS[1], skal finne objekt geometrien til et måleobjekt.

Vi ønsker å takke controlteam AS som bedrift for å ha gitt oss muligheten til å utvikle et reelt system som kanskje skal brukes. I tillegg til å ha stilt opp med alt nødvendig utstyr, samtidig som vi har fått muligheten til å jobbe sentralt hos dem. Spesielt takk til Trond G. Iversen som under hele oppgaven har veiledet oss, og passet på at vi alltid arbeidet i riktig retning. Oppgaven har vært stor og utfordrende, derfor har dette vært til stor hjelp.

Videre vil vi også takke Geir Omar Berland, Aleksandrs Mesnajevs og Olav Sande for hjelp til lisenser for nødvendig programvare. Det har blitt stilt strenge krav til programvare og versjon. Alle har vært til stor hjelp slik at vi fikk tak i riktig programvare med riktig versjon.

Vi ønsker også å takke vår veileder Tom Kjøde som har vært til stor hjelp underveis. Tom har bidratt med innspill, kontroll på arbeidsmengde og akademisk veiledning.

Sammendrag

Denne bacheloroppgaven har gått ut på å utvikle et system som ved hjelp av en PLS skal håndtere data fra en LiDAR-sensor, samt finne ut av om det er hensiktsmessig å bruke en LiDAR i samspill med PLS for å kjenne igjen objekt geometrier. Prosjektet ble gjennomført uten detaljert kunnskap om den faktiske applikasjonen til systemet, men med klare instruksjoner om oppgavens mål. Formålet har vært å gjenkjenne geometrien til et måleobjekt ved hjelp av sensordataene. Etter måleobjektet er detektert, skal en mekanisk arm kunne arbeide uten kollisjon i en fordypning i måleobjektet.

Løsningen involverte å opprette en rekke funksjoner i TIA-portal som samarbeidet for å oppnå ønsket funksjonalitet. Denne løsningen kan grovt sett deles inn i tre deler. Første del består av deteksjon av linjene i måleobjektet basert på punktdata fra LiDAR-sensoren. Deretter blir et forhåndsdefinert måleobjekt plassert langs de detekterte linjene, og disse blir sammenlignet med hverandre. Etter måleobjektet er funnet og riktig plassert, låses systemet, og går over til en modus der den mekaniske armen kan arbeide.

Oppgaven har vært en verdifull læringsprosess for oss, og vi har opparbeidet gode erfaringer på flere områder. Spesielt det å arbeide med en større og mer utfordrende oppgave enn det vi er vant til. Vi har i stor grad oppfylt kravspesifikasjonen og målene vi satte ved starten av prosjektet. Selv om noen deler kanskje ikke ble løst helt som planlagt, er vi som gruppe tilfredse med resultatet vi har oppnådd.

1 Innhold

1 Innledning

Dette kapittelet presenterer organiseringen av rapporten samt vår oppdragsgiver, Controlteam AS. Vi utforsker den komplekse problemstillingen vi ble presentert for, som omhandlet utviklingen av et system som kombinerer LiDAR-teknologi med en PLS for å gjenkjenne objekt geometri i industrielle settinger.

1.1 Organisering av rapporten

Vår rapport er organisert på denne måten.

Kapittel 1 handler om hvem som ga oss oppgaven, hva problemstillingen er og hva vår hovedide for løsningen var.

Kapittel 2 handler om kravspesifikasjonene til oppgaven.

Kapittel 3 handler om analyse av problemet, blant annet ulike løsninger og vurdering av verktøy.

Kapittel 4 handler om forklaringer av funksjoner og funksjonsblokker vi har laget for å løse oppgaven.

Kapittel 5 handler om realiseringen av den valgte løsningen.

Kapittel 6 handler om testene som har blitt gjort underveis.

Kapittel 7 handler om diskusjon rundt valgt løsning.

Kapittel 8 handler om konklusjonen vår.

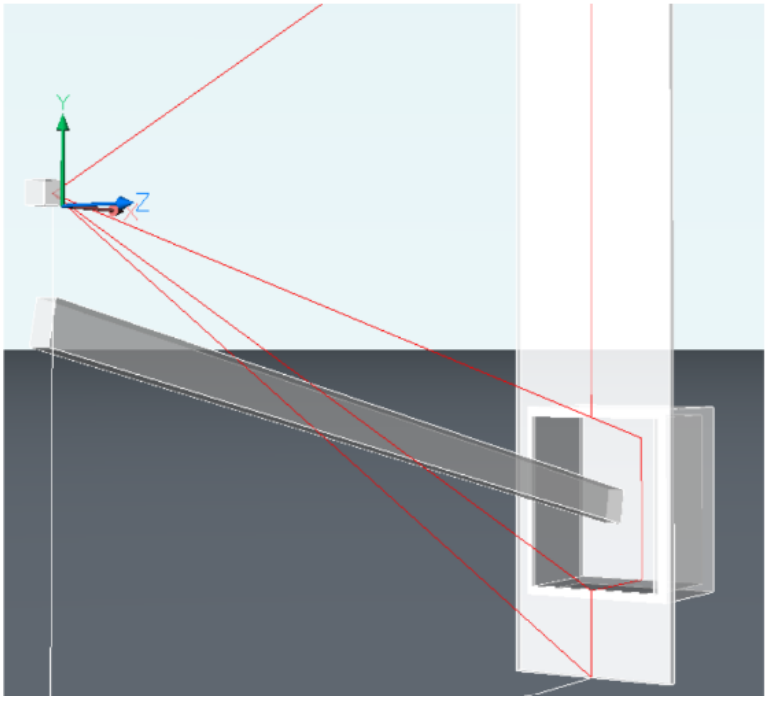
1.2 Oppdragsgiver

Controlteam AS er et ingeniørfirma som holder til på Nesttun og ble etablert i 1998. De har 17 ansatte med lang erfaring innen elektriske systemer og automasjon. Virksomheten fokuserer hovedsakelig på teknisk avanserte løsninger innen industriell automatisering. Controlteam hadde i 2022 en omsetning på 40 858 000 [1].

1.3 Problemstilling

På grunn av mulig kommersiell bruk av vår bacheloroppgave, var det reelle bruksområdet ukjent for oss. Vi manglet derfor en klar problemstilling å arbeide med, men mottok en generell beskrivelse av formålet med applikasjonen.

Applikasjonen dreier seg om utviklingen av et system som bruker LiDAR [2] teknologi i samspill med en PLS [3], for gjenkjenning av objekt geometri. Bruksfeltet til dette systemet vil være en gitt industriell setting, der et objekt blir presentert for en LiDAR-skanner. Objektet består av en flate hvor det er bygget inn et måleobjekt. Systemet skal kunne detektere denne fordypningen, slik at måleobjektet kan inspiseres ved hjelp av en mekanisk arm. I tillegg skal systemet opereres i ulike industrielle omgivelser, mulig under utfordrende værforhold.



Figur 1 - Illustrasjon av fysisk system

Fra oppdragsgiver fikk vi tilgang til konfigurerbare funksjonsblokker for kommunikasjonen mellom PLS og LiDAR. Vår oppgave var å organisere en fornuftig håndtering av data frembrakt av en LiDAR-sensor, som videre prosesseres av en PLS. Dette for at systemet kan gjenkjenne fordypninger i måleobjektet, og hindre kollisjon av en mekanisk arm som jobber. I tillegg skulle vi lage en nedskalert modell av systemet uten den mekaniske armen. Hovedfokuset var integrasjon og funksjonalitet til systemet.

Vi kunne dele opp problemet i 3 hoveddeler. Lage en algoritme for å detektere linjer i systemet, plassere ett forhåndsdefinert måleobjekt i forhold til de detekterte linjene og håndtering av eventuell kollisjon.

1.4 Hovedidé for løsningsforslag

Oppdragsgiver ønsket at vi skulle lage en nedskalert modell av den reelle applikasjonen. Deretter skulle sluttproduktet bli presentert på et HMI[1]-panel [4], der hele systemet skulle være visualisert. Det var ønsket at rådata fra LiDAR-sensor skulle vises, samt linjer tegnet av vårt system. I tillegg skulle brønnen bli plassert visuelt på systemet, og til slutt skulle systemet kunne detektere eventuell kollisjon med en mekanisk arm og systemet. Det var strenge krav til hva slags maskinvare og programvare som skulle brukes, men hvordan dette skulle oppnås innenfor gitt maskinvare og programvare, fikk vi frie tøyler til.

2 Kravspesifikasjon

Maskinvare

1. LiDAR-sensorens plassering skal stå slik at sensorens stråleplan er vertikalt på objektet. Dette er fordi den skal sees i dybde(x) og høyde (y). Vi skal ta utgangspunkt i at måleobjektet er ideelt plassert i bredden(z).
2. Brønnen skal ha en kjent form. Det vil si at vi bruker den samme formen gjennom hele prosjektet. Det er kjøpt inn skuffer som skal brukes for å illustrere brønnen og veggen til modellen.
3. For at inspeksjonen skal kunne utføres på en god og sikker måte er det vesentlig at den mekaniske armen kan operere inne i fordypningen, med tett klaring til de indre veggene, samtidig som kollisjon mellom arm og objekt skal unngås.
4. Vi skal bygge en nedskalert modell av systemet. Til dette skal det brukes 5 stk 40x30x15,5 cm eller 10 stk 30x20x13,5 cm trekasser for å illustrere veggen som skal skannes.
5. For å teste remisjonsverdiene, altså refleksjonsegenskapene til ulike fargenyanser, skal vi også kjøpe inn fargepapir for å ha på modellen. I tillegg skal vi også teste med refleks, for å se hvilken effekt dette har på remisjonen

[1] HMI: Human machine interface

Programvare

6. For kommunikasjonen mellom PLS og LiDAR skal systemet bruke funksjonsblokker som er laget på forhånd av oppdragsgiver.
7. Vi skal lage funksjonsblokker som beregner relevante koordinater og/eller vektorer. Til dette skal vi bruke måledataen vi får ifra funksjonsblokkene oppdragsgiver har gitt oss.
8. Når det skannes skal vi ha et punkt på måleobjektet som alltid skal være synlig under skanningen. Dette er fordi at dersom det er bevegelse på måleobjektet skal vi kunne følge det.
9. Vi skal lage en HMI skjerm som skal vise følgene:
    - 2D-modell av systemet
    - Rådata fra skanneren
    - Plassering av observerte geometriske former
    - Plassering av måleobjektet
    - Form av måleobjektet
    - Plassering av simulert arm
    - Form av simulert arm
    - XY-koordinater til relevante punkter
    - Margin i forhold til kollisjon i modellen
10. Løsningen skal demonstrere enn viss toleranse overfor forstyrrelser i LiDAR målingen, da oppgaven er relatert til et løsningskonsept som skal kunne anvendes i industrielle applikasjoner, inkludert utendørs operasjon under vanskelige værmessige forhold.
11. Eksport av data fra PLS tabeller og behandling av disse i Excel for å vise samsvar mellom rådata og detektert geometri.

Opsjoner

Her er noen ønskelige krav som vi kunne gjøre dersom vi fikk tid

1. Simulator kan utvides med en funksjon for å kjøre simulert arm etter en løype inn og ut av brønnen, i henhold til funksjonsbeskrivelsen for det tenkte systemet.

2. Det er ønskelig at systemet kan registrere obstruksjoner i arbeidsområdet, og at det demonstreres at systemet kan reagere dersom slike detekteres. Obstruksjoner kan eventuelt også vises grafisk i HMI.
3. Geometri for den mekaniske armen skal simuleres i PLS, og simulatoren skal tillate at armen plasseres i alle aktuelle vinkler og med aktuell utstrekning. Armens simulerte posisjon skal så evalueres mot objektet/målobjektet for å bestemme om den er i kollisjon eller ikke.
4. Simulering av arm

3 Analyse av problemet

Kravspesifikasjonene ga en relativt tydelig og rettet vei å gå for å løse problemet. I tillegg var det vanskelig å utforme alternativer for løsninger, grunnet vår manglende kunnskap om den reelle applikasjonen. Dermed var problemsstillingen hovedsakelig hvordan vi skulle få en PLS til å håndtere data fra en LiDAR sensor, for å kjenne igjen objekt geometri på en effektiv måte.

3.1 Utforming av mulige løsninger

Grunnet strenge krav til programvare og maskinvare, var det ikke ulike alternativer for dette. Derfor vil de ulike løsningsalternativene som presenteres nedenfor, fokusere på håndtering av data samt programmering av algoritmer innenfor de gitte rammene av maskinvare og programvare. Vi konstruerte da følgende alternativer for programmering.

3.1.1 Løsningsalternativ 1

En PLS er generelt ikke optimalisert for håndtering av komplekse matematiske operasjoner, på grunn av mangel av implementerte funksjoner for slike utfordringer. Grunnet at vår oppgave kan by på kompleks matematikk, har vi sett på muligheten til å programmere objekt geometri-gjenkjenningen i f.eks. MATLAB [5], for å så videre konvertere MATLAB-kode til en funksjonsblokk i TIA-Portal. MATLAB har integrerte funksjoner for å konvertere til programmeringsblokker som kan brukes i TIA-Portal [6]. Gjennom studiet har vi tilegnet oss god kjennskap til MATLAB, og det vil da ikke være nødvendig med særlig opplæring i MATLAB. Det vil derimot være nødvendig med utforskning av hvordan MATLAB kan generere programblokker til bruk i TIA-Portal.

3.1.2 Løsningsalternativ 2

Det andre løsningsforslaget går ut på å gjøre all programmeringen i TIA-Portal. Utfordringen med TIA-Portal er at det mangler innebygde biblioteker for å utføre avanserte matematiske utregninger. Derfor må vi lage egne funksjoner og funksjonsblokker som kan håndtere de matematiske utfordringene som oppstår i løpet av oppgaven.

3.1.3 Vurderinger i forhold til verktøy og HW/SW [2]komponenter

Controlteam stilte med maskinvare. Alt av maskinvare har en medfølgende programvare som er naturlig å bruke til gitt maskinvare. Dermed var det ikke mye valg for oss da det gjaldt maskinvare og programvare. Vi har under her diskutert eventuelle fordeler/ulemper, erfaringer og tid til opplæring for gitt maskinvare og programvare.

Vi fikk utdelt en LiDAR-sensor av typen OMD30M-R2000-B23-V1V1D-T-1L fra PepperlFuchs, som vist i Figur 2, [7]. Sensoren er kapabel til 250000 målinger per sekund. Dette var mer enn nok for vårt prosjekt siden vi uansett brukte lavest mulig frekvens på målingene med tanke på prosessering i PLS.

Figur 2 - Bilde av LiDAR-sensoren vi har fått utdelt, og bruker i prosjektet

Medfølgende denne sensoren finnes det en programvare for å initialisere LiDAR-sensor med flere innstillinger. Programmet PACTware brukes for denne LiDAR-sensoren, og ingen av oss hadde erfaring med programvaren fra før av. Etter en kort undersøkelse så programvaren overkommelig ut. Menyene var intuitive, og det var begrenset hvor mye arbeid som skulle utføres i denne programvaren. Primært sett endret vi kun på oppløsingen til målingen, skann frekvens og filtrering. Videre fikk vi også utdelt en Siemens PLS, CPU 1512SP-1(Figur 3)

[2] HW/SW: Maskinvare og programvare

[8]. En PLS har en begrensning på CPU[3]-kraft som alle andre maskiner, men siden en PLS må kunne utføre et program innenfor en viss syklustid er det viktig at man begrenser CPU arbeid.

Figur 3 - En CPU 1512SP-1 PN Siemens PLS

En PLS er i utgangspunktet dårlig egnet for å håndtere tunge matematiske beregninger og ikke like godt egnet til å håndtere mye data på en gang som en datamaskin. En generell datamaskin vil være mye mer egnet for slik datahåndtering. En PLS er derimot mye mer robust og pålitelig i en industriell sammenheng. Uansett skulle PLS brukes i oppgaven, uavhengig av begrensningene med prosesseringskapasiteten.

For å programmere en Siemens PLS brukes TIA-Portal. Vi brukte TIA-Portal v18 som er svært lik v16 som brukt i undervisning, dermed har vi kjennskap til dette fra før. Uansett beregnet vi 10 timer opplæring fordi vi var nødt til å bruke programmet i større grad enn det som var undervist i.

For visualiseringen ble det brukt et HMI-panel. HMI-panelet som ble brukt var Siemens HMI MTP1200, Unified Comfort Panel (Figur 4) [9]. Panelet tilbyr tilpasningsmuligheter til hva som vises på skjermen og hvordan det skal presenteres. Noe som passet bra til vår oppgave med tanke på at det var frie tøyler til hvordan visningen på HMI-panelet skulle se ut.

[3] CPU: Central processing unit

Figur 4 - Siemens HMI MTP1200 Unified Comfort Panel

Alt av komponenter var satt sammen og klart til bruk hos arbeidsgiver. Det var ikke nødvendig for oss å fysisk koble opp systemet. Vi skulle kun konstruere den nedskalerte modellen av måleobjektet.

3.2 Konklusjon av løsningsalternativ

Det ville vært til fordel for beregninger og effektivitet i arbeid, å skrive gjenkjenningen av objekt geometri i MATLAB [5]. Dette grunnet MATLAB sine gode biblioteker og egenskaper til å håndtere matematikk.

En PLS vil ha begrenset prosesseringskraft. Det var dermed viktig for oss å ha god kjennskap til kode, slik at det ble enkelt å lage systemet så effektivt og prosesseringsvennlig som mulig. Grunnet mulige komplikasjoner med effektivisering av kode generert av MATLAB, valgte vi å gå for løsningsalternativ 2, der vi gjorde alt direkte i TIA-Portal. Selv om det var vanskeligere å konstruere matematiske funksjoner direkte i TIA-Portal, vil det være lettere å endre og optimalisere kode vi selv har laget fra bunn.

Vi lage derfor noen funksjoner for matematiske operasjoner til bruk i TIA-Portal. Hver enkelt funksjon vil bli forklart senere.

4 Forklaring av funksjoner og funksjonsblokker.

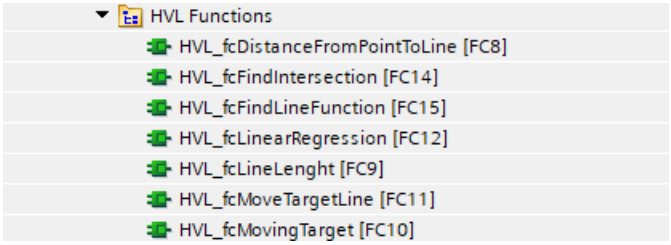
I dette kapitlet kommer vi til å forklare litt om hvordan de forskjellige funksjonene og funksjonsblokkene vi har laget fungerer. Dette kan virke litt tungt leselig, men kommer til nytte videre i oppgaven der vi skriver om hvordan vi har realisert løsningen. Der kommer vi inn på hvilke funksjoner og funksjonsblokker vi bruker for å utføre diverse oppgaver. Vi har valg å følge standard IEC61131-3 [10] når vi strukturerer og programmerer koden.

4.1 Datatyper

Vi har også definert egne datatyper, da dette gjorde det enklere å programmere det vi ønsket. Derfor lagde vi en datatype for punkt, som besto av to verdier, en x- og en y-koordinat. Dette gjorde det lettere å jobbe med punkt videre i programmeringen, og gjorde det mer oversiktlig. Lagde også datatype for linjer, som besto av x- og y-koordinater for start- og slutt-punktet på linjen, i tillegg til verdier som sa noe om linjen hadde nok punkter til å kunne defineres som en linje.

4.2 Funksjoner

For at koden skal få en bedre struktur, effektivisere gjenbruk av kode og at det skal bli enklere å feilsøke kode, lagde vi funksjoner for operasjoner vi utfører en eller flere ganger. Funksjoner er programblokker som blir kodet slik at den kan utføre alt fra enkle, til mer komplekse operasjoner. Funksjonen designes til ønsket formål, ved å bestemme hva den skal ta inn, og hva den skal returnere.



Figur 5 - Utklipp av mappestrukturen til funksjonene vi har laget i TIA-Portal

4.2.1 HVL_fcDistanceFromPointToLine [FC8]

HVL_fcDistanceFromPointToLine er en funksjon som kalkulerer den vinkelrette distansen fra ett punkt til en linje som er definert av to punkter, startpunkt og sluttpunkt. Dette punktet er ett sjekkpunkt som funksjonen får som en input, som kommer ifra der du eventuelt skal bruke funksjonen senere i koden. Funksjonen gir ut en distanse fra punktet til den gitte linjen.

Matematisk fungerer den slik at vi bruker denne formelen for å beregne distanse fra ett punkt til en linje (Formel 1) [11]. Den fungerer slik at når linjen går imellom to punkter (x1, y1) og (x2, y2), kan du finne distansen fra (x0, y0) punktet. I tillegg så ble det lagt til en boolsk variabel som skal fortelle oss om punktet ligger til høyre eller venstre side av linjen. Dette finner vi ut av ved å sjekke om telleren i (Formel 1) blir positiv eller negativ. Dette blir blant annet brukt til å sjekke kollisjon i (HVL_fbCollision [FB12]).

$$Distance = \frac{|(x2 - x1)(y1 - y0) - (x1 - x0)(y2 - y1)|}{\sqrt{(x2 - x1)^2 + (y2 - y1)^2}}$$

Formel 1 - Formel for å beregne distanse fra ett punkt som står vinkelrett på en linje

4.2.2 HVL_fcLineLength [FC9]

Denne funksjonen regner ut lengden av en gitt linje. Den får inn to punkt, et startpunkt og et sluttpunkt, og gir ut en lengde. Ved hjelp av disse to punktene, og avstandsformelen (Formel 2) regner den ut lengden av linjen. Her er «a» endring i x-verdi, og «b» er endringen i y-verdi.

$$Lengde = \sqrt{a^2 + b^2}$$

Formel 2 - Avstandsformel for å regne lengde på linje

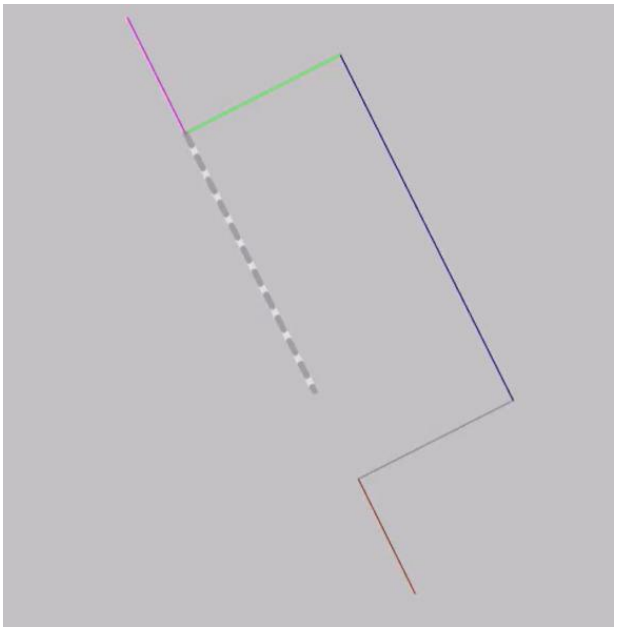
4.2.3 HVL_fcMovingTarget [FC10]

HVL_fcMovingTarget er en funksjon som plasserer hele den forhåndsdefinerte brønnen ut ifra to x-koordinater og to y-koordinater. Dette for å kunne plassere brønnen ut ifra en linje, som er plassert der vi ønsker å sette brønnen. Funksjonen starter med å finne ut av lengden og retningen til referanselinjen. Dette blir gjort ved å beregne forskjellen i x- og y-koordinatene mellom startpunktet og sluttpunktet, og brukte dette videre for å regne ut lengden av linjen ved hjelp av avstandsformelen (Formel 2).

Grunnet forhåndsdefinert brønn (Figur 6), med gitte vinkler og lengder, brukte vi trigonometri for å plassere brønnen ut ifra referanselinjen, den stiplede linjen i Figur 6. Det første vi begynte med var å lage linje 2, altså den grønne linjen i figuren. Da vi allerede hadde startpunktet på denne linjen, det samme som starten på den stiplede linjen, måtte vi bare finne sluttpunktet på linjen. Dette gjøres ved å flytte i en bestemt avstand, vinkelrett fra startpunktet til den første linjen. Må da justere x- og y-koordinatene avhengig av ønsket avstand og retning, bestemt av forskjell i x- og y-koordinatene til referanselinjen, samt lengden. Alt etter

hvilken vei linjen skal være orientert, endres justeringsfaktoren med tanke på om man bruker sinus eller cosinus. Cosinus eller sinus brukes sammen med lengden av den nye linjen, og man finner koordinater til sluttpunktet ved å projisere lengden langs de vinkelrette aksene til den opprinnelige linjen.

Etter linje 2 er laget, lages resten av linjene ved hjelp av en annen funksjon, HVL_fcMoveTargetLine [FC11]. Med da forrige linje i brønnen som referanselinje, i stedet for den stiplede linjen. Setter her inn ønsket lengde på linje, linjenummer og hvilken rotasjon som er ønsket.



Figur 6 - Figur av forhåndsdefinert brønn

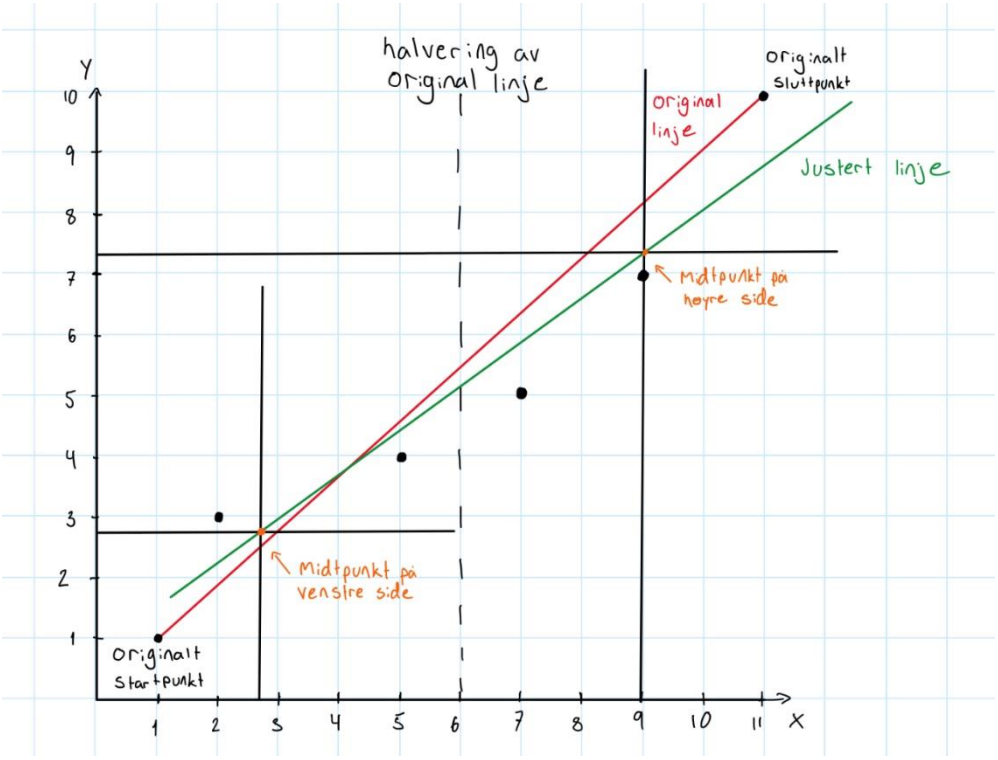
4.2.4 HVL_fcMoveTargetLine [FC11]

Funksjonen HVL_fcMoveTargetLine plasserer resten av linjene i den forhåndsdefinerte brønnen på HMI-skjermen, da HVL_fcMovingTarget [FC10] kun plasserer en linje. Denne funksjonen får inn linjenummer, lengde på linjen, og beskjed om hvordan linjen skal roteres. Dersom linjenummeret er større enn 2 blir referanselinjen den bruker for utregning linjen før, ellers er det linjen etter. Måten denne funksjonen plasserer linjene er omtrent lik som FC10, men har litt forskjeller, alt etter hvilken inndata den får.

4.2.5 HVL_fcLinearRegression [FC12]

HVL_fcLinearRegression er en funksjon som får inn to indekser «iPointnumberStart» og «iPointnumberEnd». Dette er indeksene for plasseringene av start- og sluttpunktet for en linje, i punkttabellen vår. Den starter med å finne midtpunktet mellom de to indeksene som kommer

inn. Videre er den delt opp i 4 deler, horisontalt og vertikalt for x- og y-verdiene på høyre og venstre side for midtpunktet. Alle delene gjør det samme, men den finner gjennomsnitt av antall x- og y-punkter på begge sidene av midtpunktet. Dette danner nye start- og sluttpunkt for linjen, og funksjonen vil da sende ut en ny justert linje [12].



Figur 7 -Visualisering av metode for å justere linje, lineær regresjon

4.2.6 HVL_fcFindIntersection [FC14]

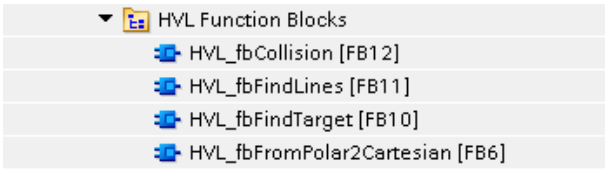
Denne funksjonen får inn to ulike linjer, og skal finne skjæringspunktet mellom disse. Bruker en funksjon vi har laget selv, HVL_fcFindLineFunction [FC15], for å finne de lineære funksjonene for linjene, $y = mx + b$. Får da ut stigningstall, og skjæringspunkt med y-aksen for begge linjene. Videre sjekker vi om disse to linjene har likt stigningstall, om de har det er de parallelle, og vil derfor heller aldri krysse hverandre. Dersom stigningstallet er ulikt, regnes det ut x- og y-koordinat for skjæringspunktet.

4.2.7 HVL_fcFindLineFunction [FC15]

HVL_fcFindLineFunction har en linje som inndata, og gir ut igjen konstantene for den lineære funksjonen for linjen, $y = mx + b$. Her er «m» stigningstallet til linjen, og «b» skjæringspunktet med y-aksen.

4.3 Funksjonsblokker

Funksjonsblokker kan i hovedsak brukes på samme måte som funksjoner, bare i tillegg så har funksjonsblokker minne til å lagre data underveis. Dette er nyttig for oss når vi skal lagre noe i en tabell eller lignende til videre bruk.



Figur 8 - Utklipp over mappe-strukturen av funksjonsblokkene vi har laget i TIA-Portal

4.3.1 HVL_fbFromPolar2Cartesian [FB6]

HVL_fbFromPolar2Cartesian er en funksjonsblokk som konverterer de polare koordinatene fra LiDAR-sensoren til kartesiske koordinater. Vi bruker følgende formel:

$$X = r * cos\theta$$

$$Y = r * sin\theta$$

Der 'r' er avstanden LiDAR måler og 'θ' er vinkelen målt. Videre filtrerer funksjonsblokken ut hvert 10. punkt av de vi har konvertert, og visualiserer disse som en punktsky på HMI-skjermen.

4.3.2 HVL_fbFindTarget [FB10]

HVL_fbFindTarget er en funksjonsblokk som forteller oss om vi har funnet brønnen i punktskyen. Det gjennomføres sjekker for å bekrefte at den forhåndsdefinerte brønnen ligger i målområdet. Den gir også ut en status på om den er funnet eller ikke.

4.3.3 HVL_fbFindLines [FB11]

HVL_fbFindLines er en funksjonsblokk som skal detektere linjer ut ifra et datasett med punkt. Funksjonsblokken itererer gjennom et sett med punkt for å sjekke om punktene kan være med å forme en rett linje. Etter den har funnet potensielle linjer, justeres disse ved hjelp av lineær regresjon. Videre finner den skjæringspunktene mellom linjene, og bruker disse til å tegne linjer mellom seg på HMI-skjermen.

4.3.4 HVL_fbCollision [FB12]

HVL_fbCollision er en funksjonsblokk som skal utføre beregninger for å håndtere eventuell kollisjon mellom den mekaniske armen og det fysiske systemet. Må ta hensyn til både

kollisjon ved selve armen mot ytterhjørnene på brønnen, og enden av armen mot alle veggene i måleobjektet.

Med tanke på kollisjon ved de ytre hjørnene på den detekterte brønnen, blir det beregnet en avstand fra hvert av hjørnene til den mekaniske armen ved hjelp av funksjonen HVL_fcDistanceFromPointToLine [FC8]. Disse to verdiene blir skrevet til en tag slik at verdiene kan vises på HMI-skjermen. Dersom en av disse avstandene blir kortere enn 5 mm, vil det være fare for kollisjon, og det vil komme en varsling på HMI-skjermen.

For å håndtere eventuell kollisjon med enden av armen til alle veggene, har vi laget en løkke for å regne ut avstanden mellom armen og alle veggene. Bruker her også HVL_fcDistanceFromPointToLine [FC8], og legger alle disse avstandene inn i en tabell. Sorterer avstandene, og viser den korteste avstanden på HMI-skjermen, da det er den som er nærmest og i størst fare for kollisjon. Dersom denne avstanden blir 0, indikerer det at det er kollisjon med en vegg, og det utløses en kollisjonsvarsling på HMI-skjermen. Tar også hensyn til om punktet på armen er til høyre eller venstre for veggene.

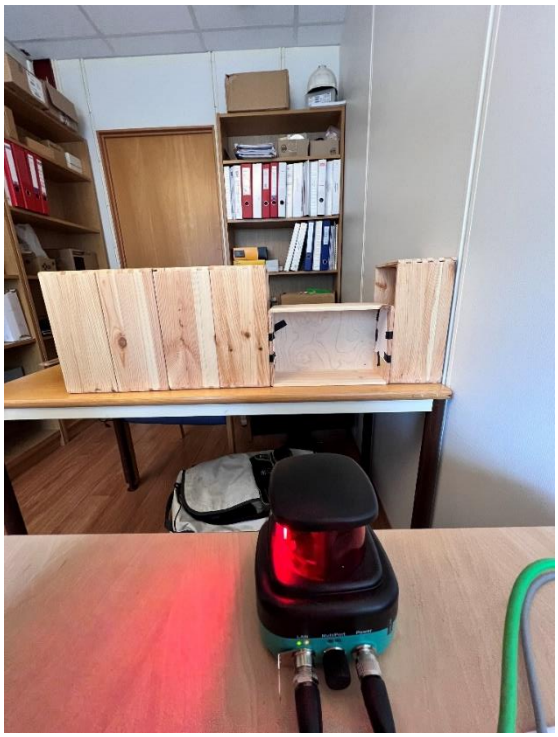
5 Realisering av valgt løsning

Tidligere har vi analysert problemet, og kommet med forskjellige løsningsalternativer. Under realisering av valgt løsning så har vi skrevet om hva som faktisk ble gjort og hvordan det ble gjort.

5.1 Nedskalert modell

For å konstruere den nedskalerte modellen av systemet brukte vi bokser fra jula [13]. Disse boksene hadde størrelsen 30x20x13,5cm, der åpningen på en boks simulerte brønnen på måleobjektet, og en sidevegg eller bakvegg simulerte en enkel vegg. Disse boksene ble satt sammen slik at vi fikk en hel vegg, med en brønnåpning et sted på veggen. (Figur 9)

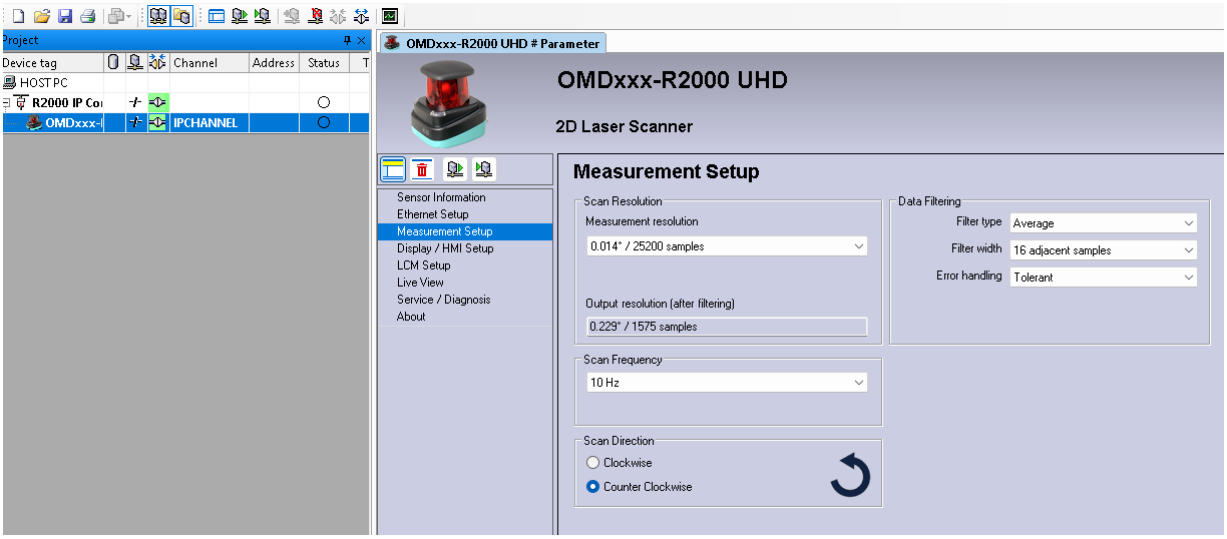
Det reelle måleobjektet skal være ca. 5 m og 3-6 m unna LiDAR-sensor. Vi har laget en modell som er ca. 1m og 0,6-1,2 m unna LiDAR-sensor. Totalt sett vil denne nedskalerte modellen være ca. 1/5 av det reelle systemet. Selv om vi ikke har en fysisk mekanisk arm som er forankret under LiDAR-sensoren, er den visualisert på HMI-skjermen.



Figur 9 - Nedskalert modell av systemet. Der måleobjektet er visualisert med trekasser, der LiDAR-sensoren står foran. Vi har vinklet systemet 90° for enkelhetens skyld

5.2 LiDAR-data

Det første vi måtte håndtere av programvare var PACTware. PACTware er programvaren tilhørende LiDAR-sensoren vi hadde fått utdelt (Figur 10). Her tok vi noen tidlige valg angående håndtering av data. Grunnet PLS sin begrensende evne til å håndtere mye data på kort tid, valgte vi å stille LiDAR-sensoren inn på lavest mulig skanne-frekvens, 10 Hz, for å få minst mulig data sendt til PLS. Dette mente vi skulle gå helt fint, da vi ikke skulle detektere et system som var i stor bevegelse. Det var derfor ikke kritisk at LiDAR-sensoren kunne detektere endringer raskt. Videre justerte vi også LiDAR-sensoren til å bruke høyest mulig filtrering. Denne filtreringen involverte gjennomsnittet av 16 punkter ved siden av hverandre. Dette gjorde vi grunnet LiDAR-sensorens unøyaktighet på ±25mm, generelle forstyrrelser og utfordringene med å filtrere data effektivt i PLS med tanke på ytelse. Selv om vi ønsket minst mulig data å håndtere for PLS, ønsket vi en nøyaktig måling. Derfor valgte vi høyest mulig oppløsning på målingene fra LiDAR. Det vil si at vi fikk veldig mange punkter målt, hver gang LiDAR-sensor gjorde en måling. Vi fikk da data fra LiDAR-sensoren som er så nøyaktig som LiDAR-sensoren kan detektere. Det ga oss muligheten til å detektere så små lengder som mulig. Til slutt endte vi opp med nøyaktige, filtrerte, men «tregt» oppdaterte målinger. Hver måling inneholdt 1575 målepunkt som oppdateres 10 ganger i sekundet. Dette vil si at vi fikk målinger med intervall på 0.229° med en frekvens på 10 Hz.



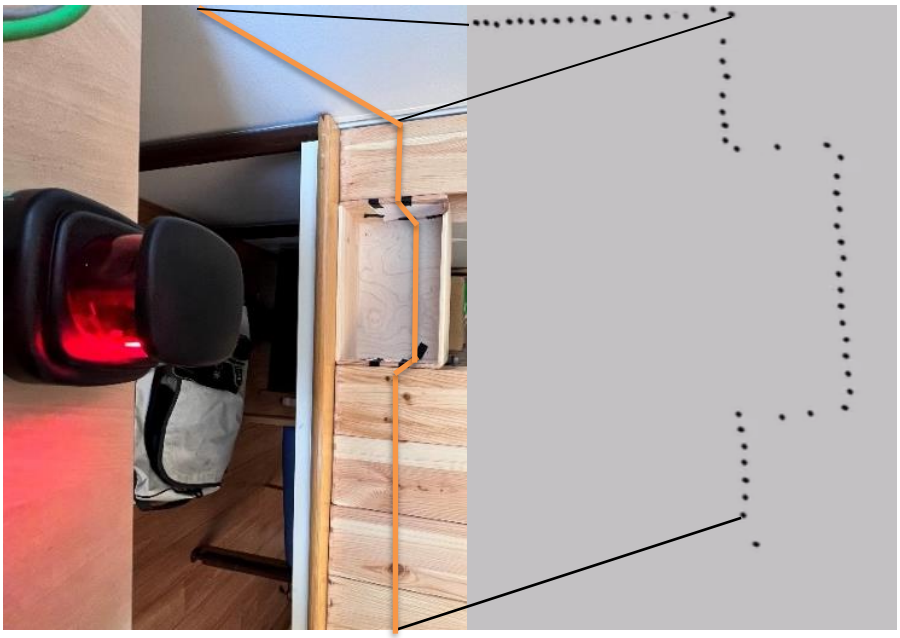
Figur 10 - Utklipp fra LiDAR-programvaren, PACTware, hvor filtreringen av målingene ble gjort

5.3 Håndtering og konvertering av data fra LiDAR i PLS

LiDAR-sensoren returnerte avstand, vinkel og remisjonsverdi for hvert målepunkt. Dermed mottok vi polare koordinater for målingene, med LiDAR-sensoren som origo, og en remisjonsverdi som vi ikke tok hensyn til før senere i prosjektet. Ettersom vårt måleobjekt besto hovedsakelig av rette linjer og rektangulære former, ønsket vi å arbeide med kartesiske koordinater. Denne omgjøringen ble gjort i en funksjonsblokk i TIA-Portal, (HVL_fbFromPolar2Cartesian [FB6]), der vi valgte ut 788 punkter fra LiDAR-skanningen. Disse punktene ble valgt slik at de spente fra -90 ° til 90°, da vi kun var interessert i det som var framfor LiDAR sensoren.

5.4 Punktsky

Punktskyen ble brukt til å vise rådata fra LiDAR-sensoren. Ut fra denne punktskyen skulle vi visuelt tegne linjer på HMI-panelet. Etter at vi hadde konvertert punktene fra polare til kartesiske koordinater (som beskrevet i avsnitt 5.2 Håndtering og konvertering av data fra LiDAR i PLS) valgte vi å ta ut hvert 10. punkt for å danne punktskyen visuelt på HMI-skjermen (Figur 11). Punktene ble lagt inn i en tabell, og på HMI-skjermen ble punktene vi la inn der, tagget til tabellen i en datablokk med disse utvalgte punktene. Disse punktene ble behandlet i samme funksjonsblokken der vi håndterte data fra LiDAR-sensoren. Dette ble gjort slik at vi kunne bruke punktene direkte fra LiDAR-sensor, for å danne punktskyen i sanntid basert på skanningen.



Figur 11 – Illustrasjon som binder punktskyen på HMI-skjermen til det fysiske systemet

5.5 Finne linjer

Etter at vi hadde behandlet data fra LiDAR-sensoren og konvertert det til et anvendelig format, var neste steg å begynne å lage linjer mellom disse punktene. Denne prosessen med å håndtere linjene ble hovedsakelig utført i vår funksjonsblokk HVL_fbFindLines [FB11]. Denne funksjonsblokken besto av flere prosesser som til slutt skulle resultere i svært nøyaktige linjer plassert riktig basert på rådataene.

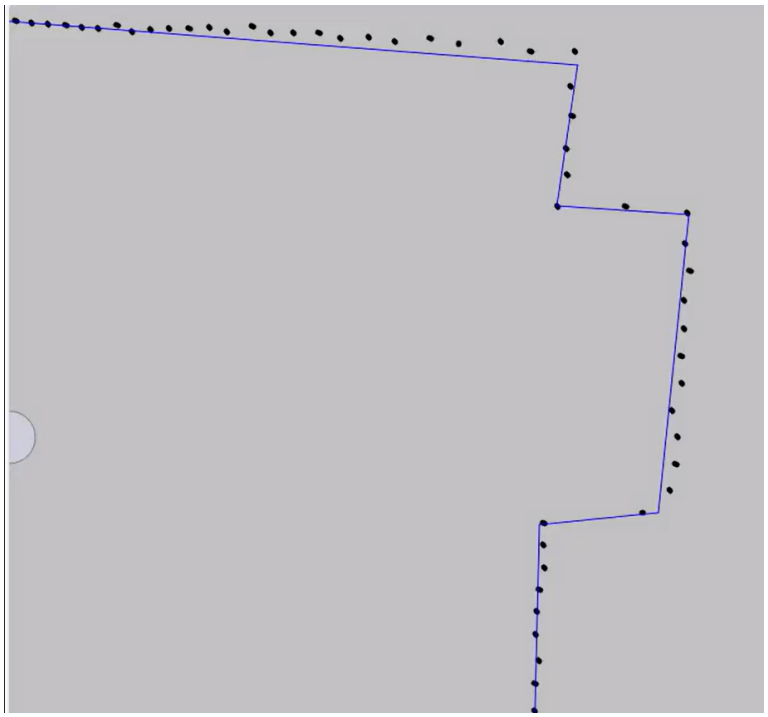
5.5.1 Original Linjer

Vi ønsket å skape linjer som var så lange som mulig. Altså at linjene strakk seg fra ett knekkpunkt, helt fram til neste knekkpunkt, (Figur 12). Vi ønsket å unngå at noe som kunne bli representert som én rett linje, ble delt opp i flere linjer. Derfor utviklet vi en løkke som skulle detektere linjene, og den genererte ti linjer. Vi hadde et kjent geometrisk utgangspunkt og fant ut at vi trengte minst fem linjer for å tegne måleobjektet. Derfor, med ti linjer, hadde vi en viss margin i tilfelle flere linjer måtte detekteres og valideres.

For å finne en linje måtte vi først initialisere start- og sluttpunktet på linjen vi ønsket å tegne. Første linje hadde startpunkt i 0, og sluttpunktet økte med et intervall på tre punkter, til linjen var ved ønsket lengde. Ingenting på det fysiske måleobjektet er kortere enn 100 mm, det vil altså aldri forekomme linjer kortere enn dette, økte derfor sluttpunktet helt til linjen ble minst 100 mm lang. Dette ble gjort i en løkke som ble kjørt så lenge lengden på linjen var under 100 mm, inne i denne løkken ble sluttpunktet økt med intervallet, og lengden ble regnet ut på nytt.

Når minimumsgrensen var nådd, kunne den faktiske linjen fortsatt potensielt være lengre, så vi kjørte en ny sjekk for å se om den kunne utvides mer før den traff et knekkpunkt.

For å sjekke om linjen kunne være lengre, utvidet vi linjen på nytt med samme intervall, og sjekket om det var punkter langs denne linjen som lå langt unna, ved hjelp av funksjonen HVL_fcDistanceFromPointToLine [FC8]. Vi hadde to sjekkpunkter, ved ½- og ¾ av linjen. Vi valgte å sjekke ved slutten av linjen, og ikke i starten, da det var der vi utvidet, og det var større sannsynlighet for at det hadde skjedd en endring der enn i starten. Begge sjekkpunktene måtte ligge langs linjen innenfor en viss margin. Hvis ikke begge punktene oppfylte kravene, gikk vi et intervall tilbake og brukte den siste godkjente linjen videre.



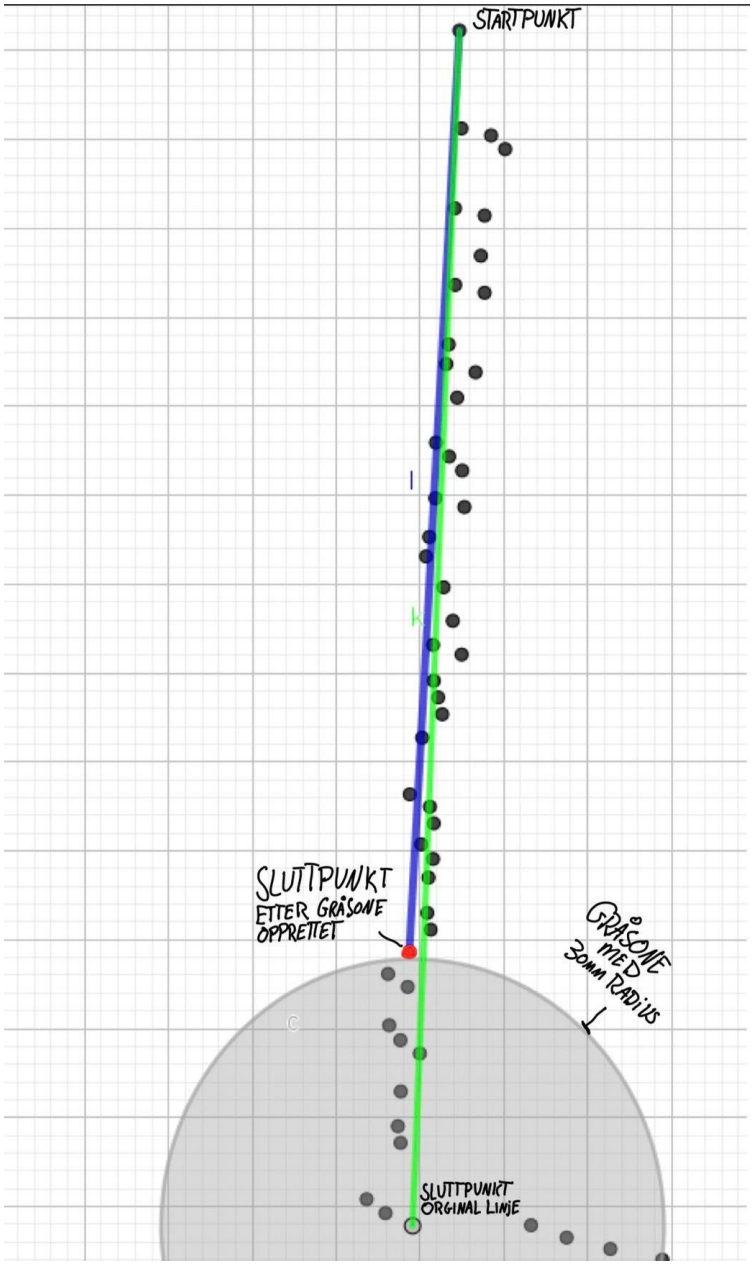
Figur 12 – Utklipp fra HMI-skjermen av detekterte linjer før justering. Første linje representerer her en vegg, og resten er deler av måleobjektet.

5.5.2 Justering av Linjer

Etter at vi hadde strukket linjene til knekkpunktene, var de fortsatt unøyaktige på grunn av at de kun var basert på start- og sluttkoordinater. Disse punktene kunne være unøyaktige i forhold til resten av den egentlige linjen. Derfor ønsket vi å utføre noe justering på de originale linjene vi hadde generert.

Det var ikke sikkert at linjene startet og sluttet på helt riktige punkter, spesielt med tanke på LiDAR-sensorens unøyaktighet på ±25mm. Derfor opprettet vi en «gråsone» rundt hvert knekkpunkt med en radius på 30 mm som vist i (Figur 13), LiDAR-sensorens unøyaktighet

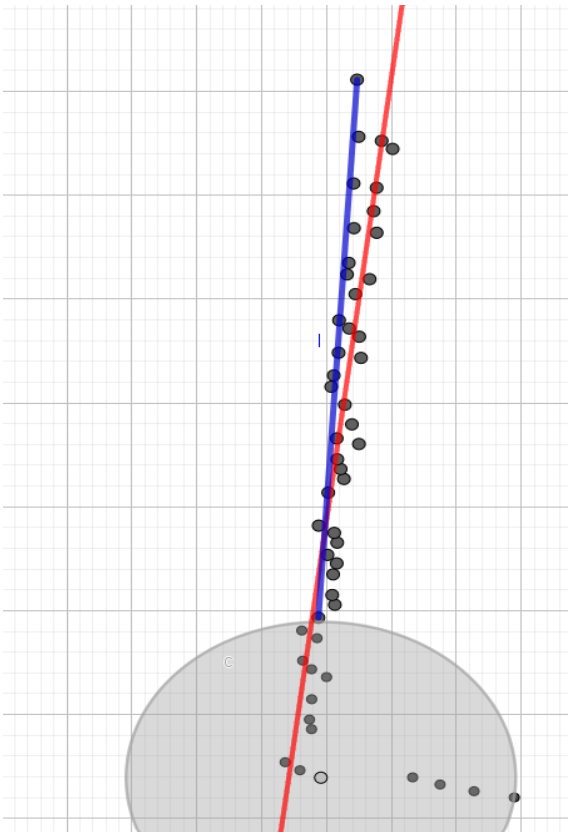
pluss litt margin. Vi opprettet denne gråsonen ved å flytte sluttpunktet på den originale linjen 30 mm bakover mot startpunktet. Gjorde dette ved hjelp av en while-løkke som flyttet sluttpunktet bakover med et intervall på 3, helt til linjen var 30 mm kortere. Vi brukte funksjonen HVL_fcLineLength [FC9] for å beregne lengden av linjen etter hver gang vi justerte den.



Figur 13 - Illustrasjon av opprettelse av gråsone, original linjen vist i grønn, og den blå linjen er linjen vi får da vi har flyttet sluttpunktet 30mm bak pga. gråsonen. Får da ett nytt sluttpunkt, vist med rød. Det grå er gråsonen med 30mm radius.

Linjene var nå fortsatt kun basert på start- og sluttkoordinater, noe som kunne være uheldig fordi vi var avhengige av at disse enkeltpunktene var nøyaktige og plassert riktig, noe som ikke kunne garanteres. Derfor kom vi frem til at lineær regresjon ville være en god tilnærming for å justere linjene i henhold til punkter mellom start- og sluttkoordinatene.

Vi brukte funksjonen HVL_fcLinearRegression [FC12], som vi hadde utviklet, der vi sendte inn indeksen til start- og sluttpunktet på original linjen. Funksjonen utførte deretter lineær regresjon ved å ta gjennomsnittet av punktene langs linjen, og returnerte deretter en linje med justerte start- og sluttkoordinater. Disse nye koordinatene var rene XY-verdier og ikke lenger indekser til punktene fra LiDAR-sensoren. Dette er vist i Figur 14, der den blåe linjen er original linjen med sluttpunkt trukket tilbake 30mm og den røde linjen er linjen som er blitt justert.

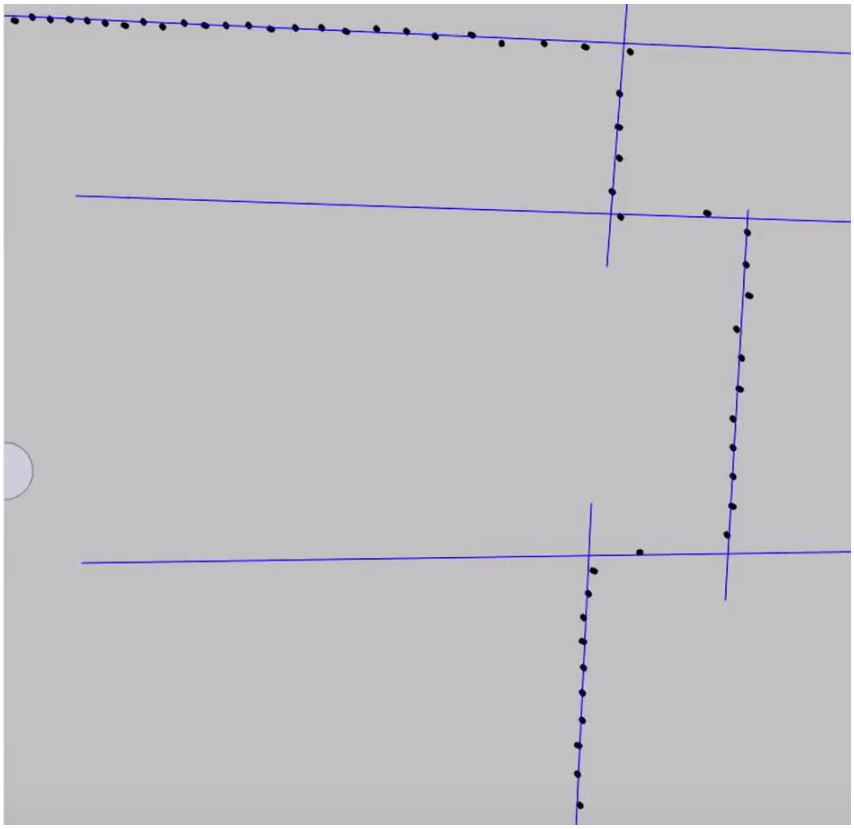


Figur 14 - Illustrasjon av linjene etter de er justert med lineær regresjon. Der den justerte linjen er den røde linjen og den blå linjen er før den blir justert. Punktene er rådata fra LiDAR-sensoren, er dermed ikke kun en tiende-del av punktene som er visualisert her.

5.5.3 Krysningspunkt

Når vi kom så langt at linjene våre var rette, gjensto det å få de til å henge sammen og ha definerte hjørner. Løsningen på det var å finne krysningspunktene mellom linjene våre, (Figur 15).

Etter at vi hadde justert linjene våre, brukte vi funksjonen HVL_fcFindLineFunction [FC15] for å få en funksjon på linjen. Når vi hadde alle funksjonene til alle linjene, brukte vi funksjonen HVL_fcFindIntersection [FC14] får å finne krysningspunktene mellom to linjer. Disse brukte vi til å sette start- og slutt-punkt på linjene, slik at vi fikk sammenhengende linjer som ikke går lenger enn de skal. Koordinatene til krysningspunktene er også visualisert på HMI-skjermen.



Figur 15 - Utklipp fra HMI-skjermen, der linjene er utvidet slik at man visuelt ser krysningspunktene

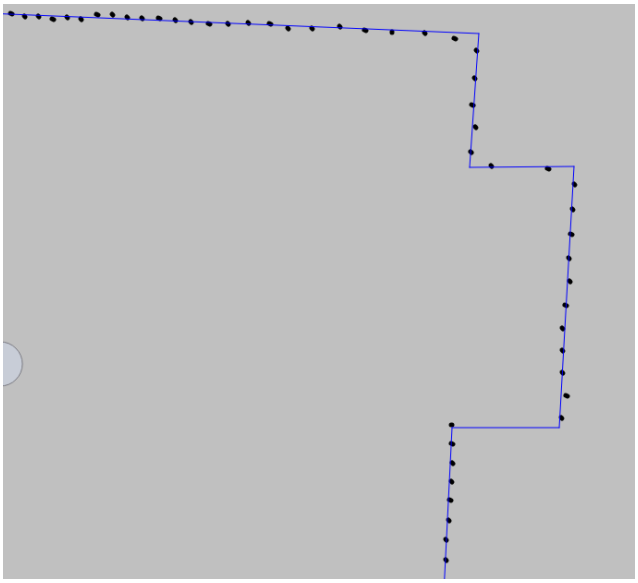
5.5.4 Tegne Linjer

Når vi skulle tegne linjene i HMI-skjermen måtte vi først i HMI-skjermen legge inn linjer vi ønsker å gi en verdi. Disse verdiene ble brukt som start og slutt punkt til en linje. Det vil si en x og y verdi for start og slutt. Hver av verdiene vi hadde funnet tidligere ble knyttet opp mot

en «tag[4]». Den «taggen» holder på en verdi slik at vi kan bruke det videre. Siden den første linjen vår ikke vil ha noe spesifikt startpunkt, valgte vi å bruke de forlengende linjene vi har funnet tidligere som startpunkt, og krysningspunktet som sluttpunkt. Det samme gjaldt den siste linjen, den har ikke noe klart sluttpunkt så vi brukte den forlengede linjen som sluttpunkt, og krysningspunktet som startpunkt. For de resterende linjene ble verdiene for start- og slutt-punkt satt til krysningspunktene mellom linjene, (Figur 17).

```
//
//tagging values to visualisation lines
//

IF #TargetFound = FALSE THEN
    // Statement section IF
    //tagging values to line 1
    #sLine[1].X1 := #slExtendedRLine[1].X1;
    #sLine[1].Y1 := #slExtendedRLine[1].Y1;
    #sLine[1].X2 := #spIntersection[2].X;
    #sLine[1].Y2 := #spIntersection[2].Y;


    //tagging values to all lines except the first and last one
    FOR #siIntersectionLinesCounter := 2 TO 5 DO
        // Statement section FOR
        //
        #sLine[#siIntersectionLinesCounter].X1 := #spIntersection[#siIntersectionLinesCounter].X;
        #sLine[#siIntersectionLinesCounter].Y1 := #spIntersection[#siIntersectionLinesCounter].Y;
        #sLine[#siIntersectionLinesCounter].X2 := #spIntersection[#siIntersectionLinesCounter + 1].X;
        #sLine[#siIntersectionLinesCounter].Y2 := #spIntersection[#siIntersectionLinesCounter + 1].Y;

        ;
    END_FOR;

    // tagging values to last line
    #sLine[6].X1 := #spIntersection[6].X;
    #sLine[6].Y1 := #spIntersection[6].Y;
    #sLine[6].X2 := #slExtendedRLine[6].X2;
    #sLine[6].Y2 := #slExtendedRLine[6].Y2;
    ;
END_IF;
```

Figur 16 - Utklipp av kode fra HVL_fbFindLines funksjonsblokken, som viser hvordan linjene i HMI-skjermen får sine verdier.

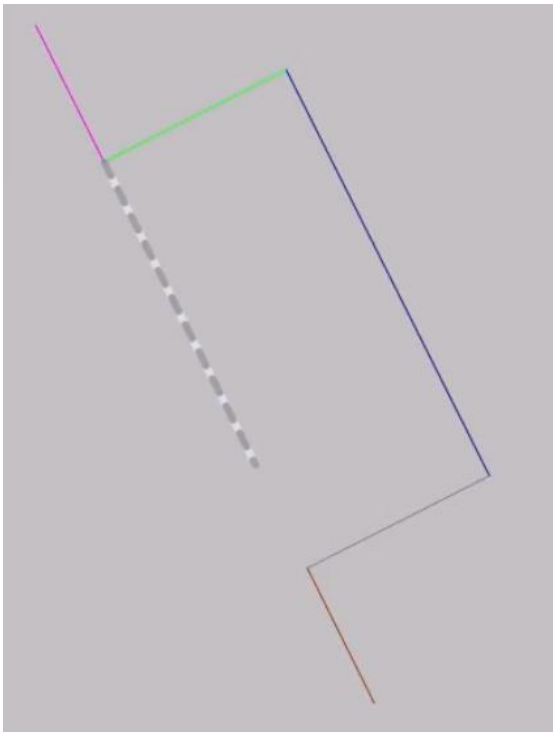


Figur 17 - Utklipp av de ferdig detekterte linjene av måleobjektet

[4] En variabel som representerer data i programvare

5.6 Plassere brønn på måleobjekt

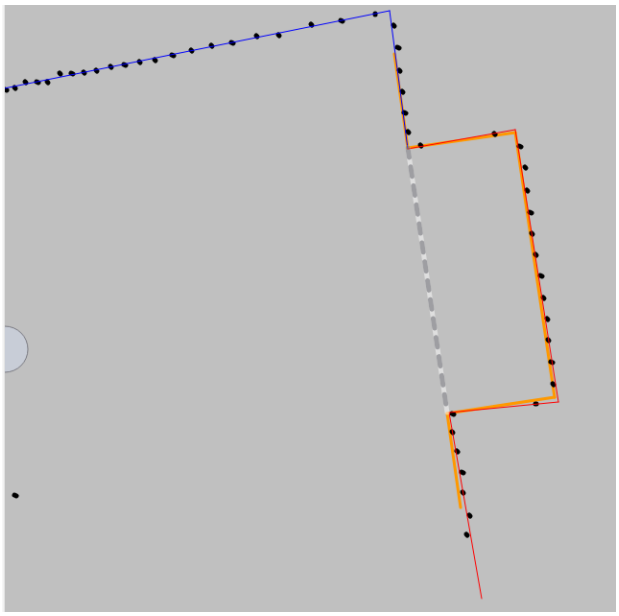
Den ene delen av oppgaven vår besto som sagt tidligere av å plassere en forhåndsdefinert brønn i henhold til det detekterte måleobjektet. Vi startet med å flytte brønnen ut ifra en referanselinje (stiplet linje) på HMI-skjermen som vi kunne legge inn start- og sluttkoordinater for. Dette gjorde vi for å kunne lage en algoritme, HVL_fcMovingTarget [FC10], som plasserte hele brønnen ut ifra en gitt linje, eller to ulike koordinater. For å kunne lage denne algoritmen, benyttet vi oss av de trigonometriske egenskapene til en rettvinklet trekant, og valgte et punkt på brønnen som skulle sammenfalle med startpunktet på referanselinjen. Trigonometrien gjorde det mulig for oss å beregne retningen og lengden av den neste linjen basert på informasjonen om den forrige linjen, og deretter projisere den gitte lengden i riktig retning for å finne sluttpunktet til den nye linjen. Til hver enkelt linje brukte vi funksjonen, HVL_fcMoveTargetLine [FC11], og sammen plasserte disse to funksjonene brønnen ut ifra to todimensjonale punkt, (Figur 18)



Figur 18 - Figur av brønn plassert ut ifra en referanselinje

Etter hvert som de detekterte linjene våre ble mer pålitelige og stabile, begynte vi å implementere funksjonsblokken, HVL_fbFindTarget [FB10], for å plassere brønnen i forhold til det detekterte måleobjektet. Dette gjorde vi ved å plassere referanselinjen (stiplet linje) langs de to ytterkantene av den detekterte brønnen (Figur 19), og deretter brukte vi samme tilnærming som tidligere for å plassere resten av brønnen. De blå og røde linjene representer

de detekterte linjene, mens de oransje representerer brønnen. For å verifisere at brønnen er plassert på korrekt sted, så har vi med ulike krav for å passe på dette.



Figur 19 - Figur av forhåndsdefinert brønn i forhold til detekterte linjer

Et av kravene for å verifisere korrekt plassering består av å sjekke lengden på linjen fra ett knekkpunkt til tre knekkpunkt fremover. Dette skyldes at vi vet lengden på åpningen av brønnen, og at den nedre delen av åpningen er tre knekkpunkt etter den øvre. Vi har derfor implementert en iterasjon som sjekker lengden mellom to knekkpunkt og sammenligner denne med den kjente lengden, med en viss toleranse for å ta hensyn til nøyaktigheten til LiDAR-sensoren.

Dersom lengden er innenfor grensene, vil koden gå videre til å sjekke vinkelen på linjen. Dette gjør vi for å finne helningen på brønnen, da vi har fått oppgitt fra oppdragsgiver at brønnen kun skal ha en helning på maks ± 10 grader. I tillegg til dette sjekker vi også om bakveggen i brønnen er tilnærmet lik parallell med referanselinjen, for å verifisere at det faktisk er brønnen vi har detektert. Om både helningen på brønnen er under 10°, og bakveggen og referanselinjen er omtrent parallelle, vil brønnen plasseres på de detekterte linjene ved hjelp av funksjonen HVL_fcMovingTarget [FC10].

Det vil i tillegg bli satt en boolsk verdi som gir en status på om måleobjektet er funnet eller ikke. Dersom denne verdien blir sann, vil løkken stoppe, og brønnen vil stå fast plassert helt til betingelsen blir usann. Samtidig vil oppdateringen av de detektere linjene på HMI-skjermen også stoppes, slik at et stillbilde av deteksjonen kan bevares. Etter løkken, sjekker

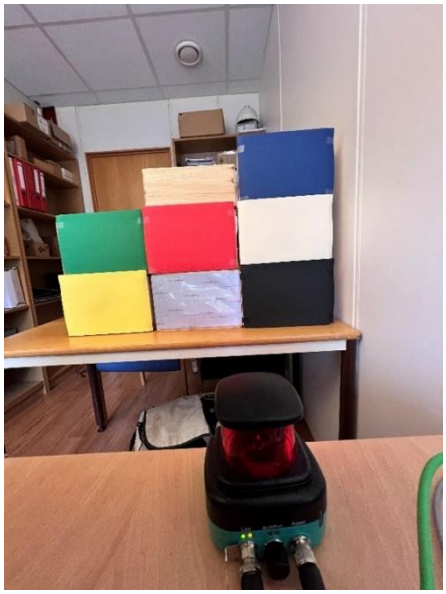
koden om målet har beveget seg mer enn en viss margin i forhold til toppunktet i brønnen, da dette alltid skal være synlig. Dersom dette skjer, vil den boolske verdien for om måleobjektet er funnet, bli satt til usann, og løkken vil starte igjen for å plassere brønnen på nytt. Linjedeteksjonen vil også starte igjen.

5.7 Eksportering av data for sammenligning

For å sammenligne rådata fra LiDAR-sensoren med ferdig detektere linjer av vårt system var det noe data vi måtte eksportere. Dette løste vi ved å hente ut alle originale koordinater fra de kartesiske koordinatene fra LiDAR-sensoren og lime inn i Excel. Dermed hentet vi ut kun relevante data og lagde en tabell av xy-koordinater. Denne tabellen limte vi inn i Geogebra [14] for å visualisere alle punkter som LiDAR-sensoren har målt. Deretter hentet vi ut alle krysningspunktene til de detekterte linjene av vårt system. Disse krysningspunktene gikk gjennom samme prosedyre som de originale koordinatene for å bli visualisert i Geogebra. Til slutt kunne vi visuelt sammenligne hvor våre genererte krysningspunkt legger seg i henhold til originale målepunkt fra LiDAR sensoren.

5.8 Remisjon

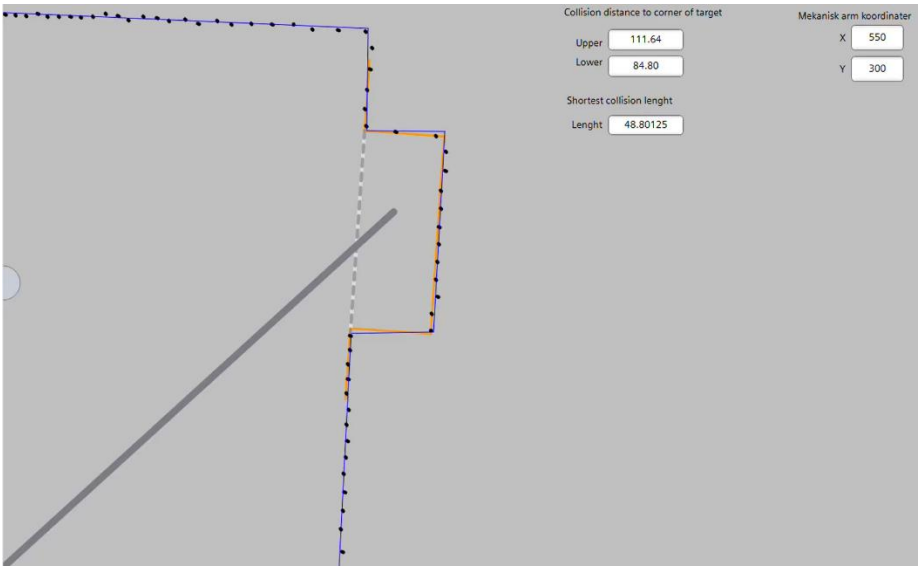
En del av oppgaven gikk ut på å teste remisjonsverdi på ulike fargeoverflater og avstander. Remisjonsverdi er en verdi på hvor godt et objekt returnerer lys fra LiDAR-sensoren, altså hvor god refleksjonsverdi noe har i henhold til LiDAR-sensoren. For å teste remisjonsverdi gikk vi vil innkjøp av fargepapir [15]. Disse fargepapirene teipet vi på trekassene vi hadde for den nedskalerte modellen av systemet. Vi hadde også en boks med refleksteip [16] for å teste refleksjonsverdien på refleks. Da kunne vi med en og en farget boks om gangen, lese av remisjonsverdien ved forskjellige avstander, ut ifra datablokken som inneholder data fra LiDAR-sensoren.



Figur 20 - Bilde av de ulike fargene og overflatene vi testet remisjon på

5.9 Mekanisk arm og kollisjonshåndtering

For å simulere den mekaniske armen valgte vi for enkelhets skyld å plassere en enkel linje på HMI-skjermen. Denne armen hadde et startpunkt som ble forankret i nedre venstre hjørne. Sluttpunktet til armen ble knyttet til en tag. Denne taggen ble knyttet til to tekstbokser på HMI skjermen der vi kunne fylle inn x- og y-koordinat. Da flyttet linjen sitt sluttpunkt seg til det ønskede koordinatet (Figur 21).



Figur 21 - Utklipp fra HMI-skjermen som viser at armen er plassert ut ifra X- og Y-koordinater som er plassert øverst i høyre hjørne

For å vise at systemet kunne håndtere kollisjon mellom den mekaniske armen og linjene, har vi tatt noen valg. Siden på armen kan kun kollidere med de øvre og nederste ytterhjørnene på

brønnen (Figur 22). Derfor beregnet vi avstanden fra disse to hjørnene til den mekaniske armen, og visualiserer disse verdiene. For å utføre dette brukte vi funksjonen HVL_fcDistanceFromPointToLine [FC8]. Fare for kollisjonen blir vist på skjermen med en rød firkant med «WARNING COLLISION!!!!», dette skjer når lengden fra det øverste hjørne eller det nederste hjørne til armen er mindre enn 5 mm.

Figur 22 - Utklipp fra HMI-skjermen der du ser at armen har kollidert med det nederste ytrehjørne

Det er også en mulighet at tuppen av armen kan kollidere med veggene av måleobjektet, det er derfor laget noen sjekker. Måten den sjekker det på er at vi igjen bruker funksjonen HVL_fcDistanceFromPointToLine [FC8] der vi sjekker avstanden fra punktet (tuppen av armen) til linjen (veggen). I funksjonen får vi også ut en boolsk variabel som forteller oss om punktet ligger til høyre eller venstre for linjen. Dette brukes videre til å vurdere om det er kollisjon eller ikke. Hvis den boolske verdien sier at punktet er på høyre side av linjen og lengden fra punktet til linjen er lik null, er det kollisjon (Figur 23).

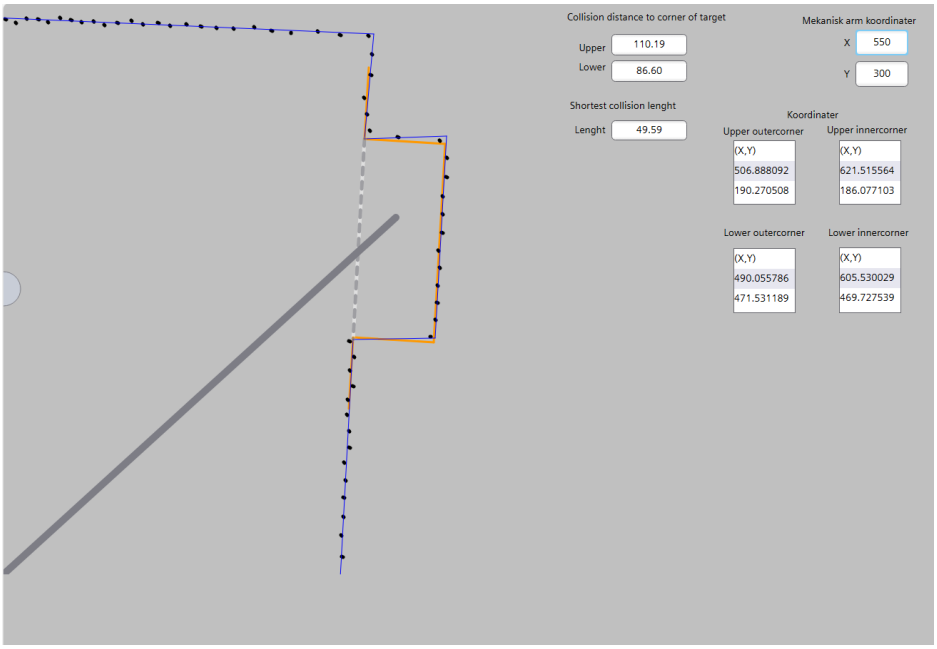
Figur 23 - Utklipp fra HMI-skjermen av at enden på armen kolliderer med den bakre veggen av brønnen.

5.10 HMI-brukergrensesnitt

En del av oppgaven var å visualisere resultatene våre i et HMI-panel. Til dette har vi fulgt kravspesifikasjonene vi fikk utdelt i oppgaven. Her kommer en forklaring på hva som er visualisert i (Figur 24).

Det vi startet med var å visualisere rådata fra LiDAR-sensoren i form av en punktsky, dette er de svarte prikkene på venstre halvdel av skjermen. De blå linjene representerer de detekterte og justerte linjene, mens de oransje linjene er det forhåndsdefinerte måleobjektet som skal sammenlignes med de detekterte linjene. Videre på venstre halvdel av panelet så ser dere en liten halvsirkel, dette skal illustrere LiDAR-sensorens plassering i forhold til måleobjektet. Under den er den mekaniske armen plassert i henhold til kravene for plassering av arm i reelt system.

På høyre halvdel av panelet er de relevante dataene plassert i form av IO-felt. Her viser vi blant annet avstander i forhold til mulige kollisjoner. For at vi skulle kunne bevege armen, har vi lagt inn to felt øverst i høyre hjørne, der vi kan sette koordinatene til plasseringen av enden av armen. Til slutt viser vi også relevante koordinater som hjørnene i måleobjektet.



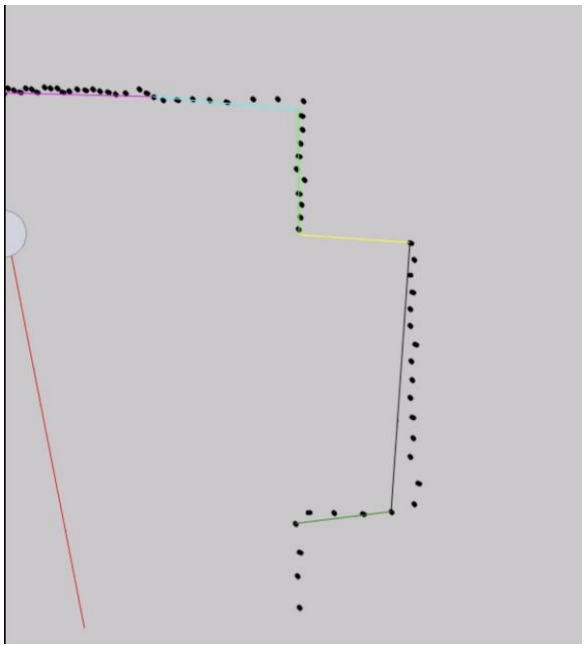
Figur 24 - Visualisering av realisert løsning på HMI-skjerm

6 Testing

Vi har utført noen enkle visuelle tester av algoritmer underveis for å kontrollere arbeidet. Hovedsakelig vil testene bestå av linjer generert underveis, sammenlignet med punktsky. Punktskyen er en visualisering av hvert tiende punkt fra LiDAR-sensor. Under vil du se visuelt hvordan dette er utført.

6.1 Test av originale linjer.

Dette var en test utført før vi skapte linjer på minimum 100mm. Vi har her utvidet linjene så langt som mulig helt til et punkt på linjen var for langt unna, (Figur 25). Linjene var relativt ustabile og lå ikke helt nøyaktig på punktene. Ut ifra denne testen kom vi fram til at vi må lage linjer med minimumslengde på 100 mm for å unngå at noe som kan bli representert som en linje, blir representert ved to eller flere. I tillegg til at vi også var nødt til å justere linjene.



Figur 25 - Linjer som blir strukket før justering

6.2 Test av remisjonsverdi

Som en del av oppgavebeskrivelsen var det ønskelig å teste remisjonsverdien til ulike farger på måleobjektet ved ulike avstander. En remisjonsverdi er en tallverdi på hvor god refleksjon av LiDAR-laseren et objekt har. For å utføre testen brukte vi de samme trekassene som i den nedskalerte modellen, men vi limte på forskjellige fargepapir og refleks.

Før vi startet testen så satte vi oss ned og formulerte en hypotese. Basert på våre tidligere observasjoner og teoretiske kunnskaper forventet vi at remisjonsverdien til LiDAR-sensoren ville variere avhengig av fargen på bakgrunnen, med en forventning om at noen farger ville reflektere bedre enn andre. Vi antok at treverket som en naturlig og ujevn overflate ville reflektere LiDAR-laseren dårligere enn noen av de ensfargede fargene på grunn av sine materialegenskaper. Hypotesen vår var også basert på antakelsen om at refleksjonsegenskapene til de forskjellige fargene på bakgrunnen ville påvirke sensorens evne til å registrere objektene nøyaktig. Vi forventet derfor at fargene med høyere refleksjonsverdier ville gi mer pålitelige og nøyaktige LiDAR-målinger, mens treverket kunne mulig forstyrre nøyaktigheten til sensorens avlesninger på grunn av lavere refleksjonsverdier og ujevnheter i overflaten.

Vi fikk da følgende resultat (høyere remisjonsverdi er bedre):

Farge/Avstand[mm]	600	1200	2000	4000
Tre	431	535	414	278
Refleks	1155	1294	1221	1128
Svart	377	479	367	272
Hvit	392	494	384	275
Rød	389	484	376	275
Blå	379	476	367	271
Gul	386	481	375	272
Grønn	380	479	370	272

Tabell 1 - Tabell av remisjonsverdiene som vi fant under testing av remisjon, der rød er svak refleksjon og grønn er god refleksjon

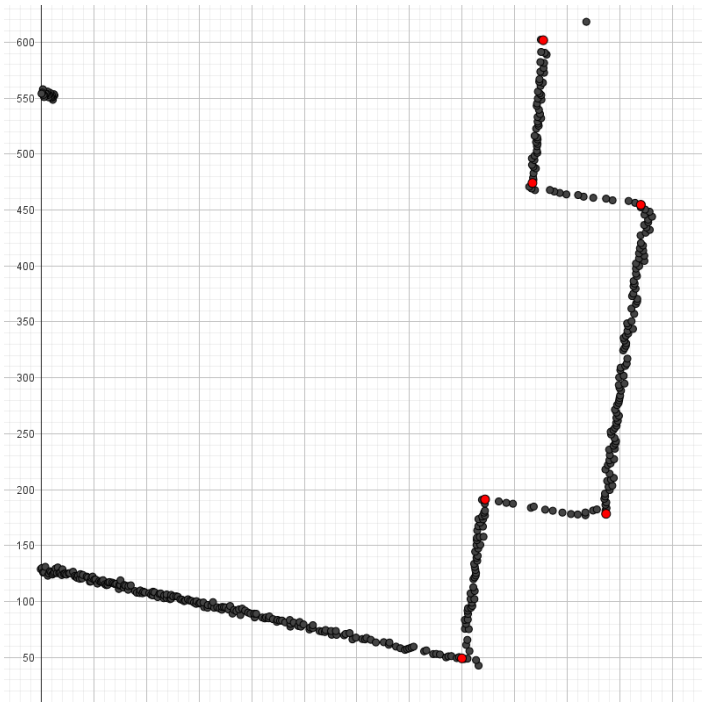
Våre resultater viser at fargene alene ikke hadde en betydelig innvirkning på remisjonsverdiene, noe som utfordrer vår opprinnelige hypotese om at visse farger ville reflektere bedre enn andre. Det viste seg imidlertid at avstanden hadde noe å si i forhold til remisjon, de beste resultatene kom på avstand på 1200mm.

En overraskende observasjon var at fargene reflekterte dårligere enn treverket. Dette står i kontrast til vår opprinnelige forventing om at treverk, som en naturlig og ujevn overflate, ville reflektere dårligere enn de ensfargede fargene på grunn av sine materialegenskaper. Det er mulig at denne uventede observasjonen er knyttet til spesifikke egenskaper ved fargepapirene eller andre faktorer vi ikke tok hensyn til i våre tester.

Selv om vi så små forskjeller mellom fargene kan vi se at lyse farger som hvit reflekterer litt bedre enn mørke farger, som svart og den mørkeblå fargen vi brukte. Denne observasjonen indikerer en generell trend der lysere farger har en tendens til å ha høyrere refleksjonsverdier sammenlignet med mørkere farger. Den relative refleksjonsegenskapen til farger kan være knyttet til deres evne til å absorbere eller reflektere lys, hvor lysere farger har en tendens til å reflektere mer lys og mørkere farger har en tendens til å absorbere mer lys.

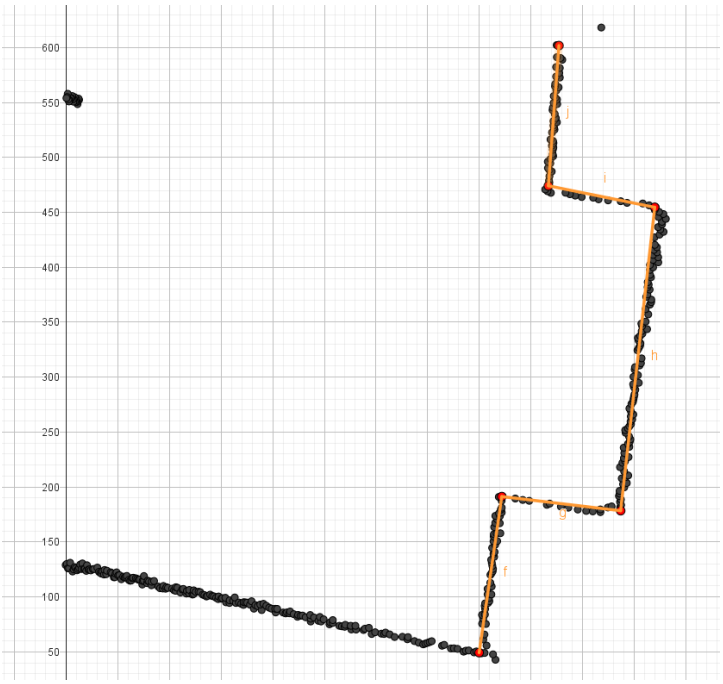
6.3 Test av genererte linjer og rådata

Som nevnt tidligere, var en del av oppgaven vår, å eksportere relevant data fra PLS-tabeller for å sammenligne rådata med detektert geometri. Vi benyttet Excel og Geogebra for å utføre denne oppgaven, og gjennom disse verktøyene kunne vi visualisere hvor nøyaktig vår deteksjon var. For å representere rådata, valgte vi å eksportere alle de originale koordinatene vi fikk fra LiDAR-sensoren, og for den detekterte geometrien eksporterte vi krysningspunktene vi hadde detektert, (Figur 26). Etter å ha visualisert disse i lag, kunne vi se at de detekterte krysningspunktene la seg stort sett på riktig plass. Det var derimot noe større avvik i krysningspunktene ved hjørnene inne i brønnen. Vi antar at dette kan skyldes en kombinasjon av dårlig synsvinkel og linjer som er dannet med færre punkter, noe som igjen resulterer i mindre justeringsmuligheter for linjene.



Figur 26 - Visualisering i Geogebra av sammenligningen av rådata og detektert geometri

Ved å strekke linjer mellom krysningspunktene (Figur 27), kunne vi representere det detekterte måleobjektet i forhold til rådataen fra LiDAR-sensoren. Dette tillot oss å observere at visualiseringen av den detekterte brønnen har klare likheter med rådataen.



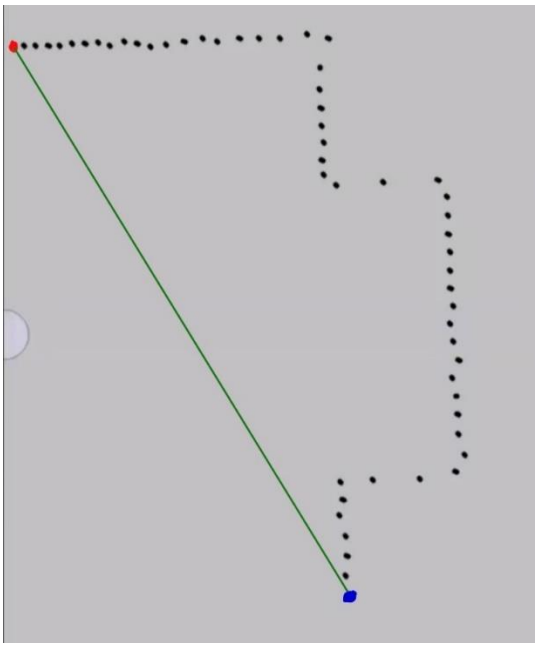
Figur 27 - Visualisering i Geogebra av detektert måleobjekt, mot rådata fra LiDAR-sensor

7 Diskusjon

I dette kapitlet kommer vi inn på diskusjon om valgene vi har tatt. Det vil si, sammenligne det som er gjort gjennom prosjektet, hva som er gjort feil, hvilke valg vi har gjort i forhold til forprosjektet og hva vi ville gjort annerledes.

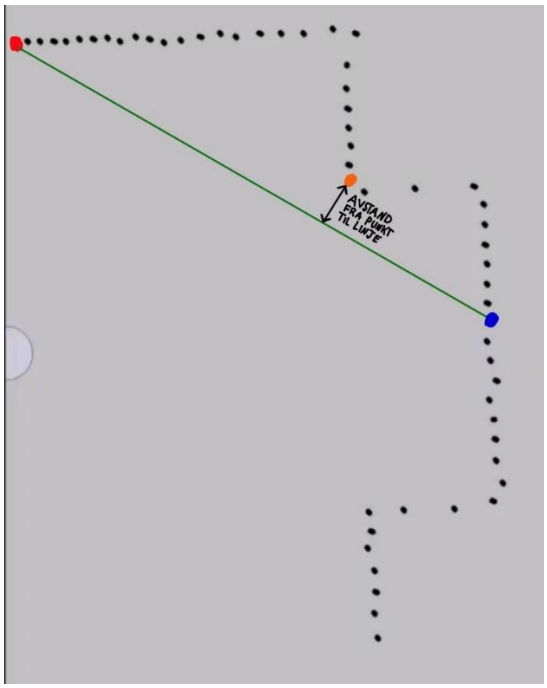
7.1 Originale linjer

I prosessen med å tegne de originale linjene var vi innom ulike metoder. Løsningen vi endte opp med, som sagt tidligere i (5.5 Finne linjer), var å begynne med korte linjesegmenter, og deretter øke sluttpunktet med et forhåndsbestemt intervall inntil linjen oppfylte kriteriene vi sjekker. En annen måte å gjøre dette på var å starte med en linje som strakte seg fra det første punktet(rødt), til det aller siste punktet(blått), som vist i (Figur 28). Deretter gradvis redusere sluttpunktet med det angitte intervallet slik at sjekkpunktene imellom var innenfor akseptabel avstand til linjen. Etter dette ble det nye sluttpunktet brukt som startpunktet for neste linje, mens sluttpunktet på den nye linjen ble igjen satt til det aller siste punktet og algoritmen startet igjen.



Figur 28 - Visualisering av hvor linjen starter før den minker med bestemt intervall. Startpunktet av linjen er rød, og sluttpunktet er blå.

En utfordring med denne løsningen var at den i noen tilfeller ville tegne linjer der det ikke skulle være linjer. Dette skjedde for eksempel når hjørnene på måleobjektet ble brukt som sjekkpunkter, og algoritmen feilaktig antok at den hadde funnet en rett linje på riktig sted, som vist i Figur 29.



Figur 29 - Visualisering av et tilfelle der sjekkpunktet, vist i oransje, havner på et hjørne, og fører til at avstanden mellom punkt og linje går innenfor marginen, og dermed blir linjen plassert på feil plass.

Vi tok også et viktig valg angående antall sjekkpunkter som skulle brukes ved utvidelsen av linjene. Startet først med tre sjekkpunkt på ¼, ½, og ¾ av linjelengden, men fant senere ut at det ikke var nødvendig med så mange. Selv om flere sjekker gir en mer pålitelig validering av linjen, fant vi ut at to sjekkpunkter var tilstrekkelig i dette tilfellet. Fjernet derfor sjekkpunktet på ¼ av linjen, da sannsynligheten for stor endring er større mot slutten av linjen enn starten. Dette viste seg å ha minimal innvirkning på resultatet, samtidig som programmets syklustid ble redusert. Ettersom de originale linjene uansett strakte seg litt for langt, uavhengig om vi hadde to eller tre sjekkpunkter, så bestemte vi oss for å kun bruke to. Problemet med at linjene gikk for langt ville bli løst når vi skulle justere linjene ved å flytte sluttpunktet 30 mm bakover for å ta hensyn til gråsonen i hjørnene. Dermed ville ikke det ha veldig mye å si om linjene strakk seg noe lenger med to sjekkpunkter i stedet for tre.

Grunnen til at vi genererte et fast antall linjer (10 linjer) var fordi vi var nødt til å lagre alle linjene i en liste. Når man lager en liste i TIA-Portal, kan man ikke ha en liste med variabel eller ukjent lengde. Man må angi hvor mange elementer det skal være i listen. Dette har med robusthet å gjøre. TIA-Portal må vite nøyaktig hvor mye minne den skal sette av for å lagre denne listen.

I flere prosesser under generering av linjer brukte vi et intervall på 3 punkter. Dette intervallet ble brukt når punkter flyttes, sjekkes eller lignende. Det ville vært mer nøyaktig å ha et intervall på 1 punkt. Da ville alle punkt vært med i prosesser som sjekker linjer. Grunnen til at vi ikke valgte et intervall på 1 punkt er på grunn av syklustiden til programmet. Det er viktig at PLS kan håndtere programmet innenfor en viss syklus tid. Med intervall på 1 fikk vi alt for høy syklustid. Det var da hensiktsmessig å øke intervallet for å minke beregninger PLS må utføre. Vi kom fram til at et intervall på 3 var fornuftig med tanke på syklustid og nøyaktighet.

7.2 Justering av linjer

Vår tilnærming for å forbedre nøyaktigheten av linjene ved hjelp av lineær regresjon har vist seg å være vellykket. Ved å introdusere en "gråsone" rundt knekkpunktene og anvende regresjon mellom start- og sluttkoordinatene, har vi kunnet filtrere ut unøyaktigheter som oppstår fra LiDAR-sensoren.

I løpet av prosessen med å utvikle denne tilnærmingen, eksperimenterte vi med ulike metoder for å finne den mest effektive tilpasningen. I begynnelsen foreslo vi at linjene skulle

begrenses til kun de første 15 mm på hver side av midtpunktet, for å deretter utføre regresjon. Mens dette resulterte i tilfredsstillende resultater, innså vi at det var en begrensning knyttet til det begrensede antallet punkter som ble inkludert. Dette kunne mulig føre til unøyaktigheter, og var vanskelig å forsvare metodisk.

Som et svar på dette, justerte vi vår tilnærming ved å forlenge linjene så langt som mulig fram til knekkpunktene, før vi trakk dem tilbake 30 mm, og deretter utførte regresjon. Dette tillot oss å ta hensyn til et større antall punkter langs linjen og produserte dermed en mer pålitelig linjeføring. Ved å ta gjennomsnittet av flere punkter, fikk vi et bedre utgangspunkt for vår lineære regresjon og kunne dermed oppnå høyere nøyaktighet i våre analyser. Vi ønsket også å flytte startpunktet på linjen 30mm mot slutten, men det er ikke utført grunnet komplikasjoner. Dermed vil linjen i prinsipp bli minimum 30mm kortere og maksimum 60mm kortere om vi hadde flyttet startpunktet også. Det går fint fordi vi i starten lager alle linjer minimum 100mm.

Om vi hadde hatt mer tid hadde vi ønsket å kjøre en form for filtrering før den lineære regresjonen. Da hadde vi gått gjennom hvert punkt på den linjen som skal justeres og sjekket avstanden fra hvert punkt til linjen. Om et punkt har en avstand større enn en terskel ville vi filtrert vekk dette punktet før vi bruker det i beregningen til lineær regresjon. Det ønsket vi på grunn av at det punktet mest sannsynlig da vil være en form for forstyrrelser eller feilmåling. Grunnet liten tid og effektivisering av PLS kode utførte vi ikke denne filtreringen.

7.3 Plassere brønn på måleobjekt

Vi har satt opp flere sjekker for å verifisere korrekt plasseringen av brønnen. En av disse sjekkene var å verifisere at åpningen i den antatte brønnen var omtrent lik de forhåndsbestemte dimensjonene vi hadde angitt. Om lengden ikke var innenfor grensene, hadde vi heller ikke funnet måleobjektet, og måtte dermed lete videre. En annen viktig sjekk var å kontrollere om bakveggen i brønnen er omtrent parallell med referanselinjen, som vil bekrefte om vi faktisk har detektert brønnen. Den siste sjekken var basert på kravet om at helningen på brønnen bare skulle være innenfor ±10°. Hvis helningen er større enn dette, ville heller ikke brønnen bli plassert på det detekterte måleobjektet.

Dersom vi hadde hatt mer tid til rådighet, kunne vi ha inkludert enda flere sjekker for å forsikre oss om at brønnen var riktig plassert. Disse sjekkene kunne blant annet vært å kontrollere at veggene før og etter brønnen var omtrent parallelle med bakveggen i brønnen,

slik at vi kunne være sikre på at det ikke bare var en tilfeldig fordypning. Vi kunne også ha vurdert å sjekke lengden fra åpningen til bakveggen, da denne lengden er gitt og må ligge innenfor en viss toleranse for å bli definert som brønn. Videre visste vi at hjørnene på brønnen skulle være vinkelrette, og kunne også brukt dette for å verifisere at brønnen var korrekt plassert.

Etter at systemet har funnet og plassert måleobjektet vil systemet slutte å lete etter ny plassering, helt til det øverste hjørnet på brønnen flytter seg. Grunnen til at vi valgte å følge med på det øverste hjørnet på brønnen, er fordi det er det eneste punket vi er helt sikre på at LiDAR-sensoren alltid vil se. Dette kan vi garantere fordi den mekaniske armen står under LiDAR-sensoren, dermed vil armen aldri kunne skygge for det øvre hjørnet når den opererer, den kan derimot skygge for samtlige andre hjørner i target.

7.4 Effektivisering

Under hele prosjektet har vi måtte tatt valg mellom å oppnå effektiv håndtering, eller nøye håndtering. Det er viktig at en PLS kan håndtere programmet effektivt uten problemer, men det er også viktig at vi lager et system som er så nøyaktig som mulig. Disse to tingene vil jobbe mot hverandre og vi har dermed vært nødt til å ta noen valg. Vi er fullt klar over at det er mulig å ha et mer nøyaktig system, men grunnet syklustiden har vi gjort noen forenklinger. Dette innebærer blant annet å minke intervall på sjekker fra hvert punkt til hvert 3. punkt. Fjerne et av 3 sjekkpunkt på linjene som genereres, og generelt prøvd å begrense beregninger som PLS må håndtere.

7.5 Måleusikkerhet

Det originale måleobjektet vil være vesentlig større enn vår nedskalerte modell. Det originale måleobjektet vil være ca. 5 m høyt, mens vår nedskalerte modell var ca. 1 m høy. Vår modell bruker samme LiDAR-sensor som det originale systemet, ulempen med dette er at LiDAR-sensoren har en fast måleusikkerhet på ± 25 mm. Det vil si at vi hadde en mye større prosentandel på måleusikkerheten, og dermed hadde vår nedskalerte modell «dårligere» målinger enn det ekte systemet vil ha. Derfor har vi også en del høye terskler til marginer.

Siden LiDAR-sensoren har en måleusikkerhet på ± 25 mm, har vi alltid tatt hensyn til dette når vi har satt terskler i koden. F.eks. når vi sjekket om en verdi er innenfor en viss terskel, satt vi alltid terskelen til minimum ± 25 mm.

7.6 Mekanisk arm og kollisjon

I vår nåværende systemimplementasjon detekterer vi kollisjoner mellom den simulerte mekaniske armen og måleobjektet. Imidlertid har vi ikke prioritert å implementere forhindring av disse kollisjonene grunnet andre prioriteringer og tidsbegrensninger.

Dersom vi hadde hatt mer tid til rådighet, ville vi ha utforsket en annen tilnærming. Istedenfor å ha en arm der vi setter koordinatene manuelt, ville vi ha utviklet en arm som beveger seg selv når systemet har detektert måleobjektet. Tanken bak dette er å implementere en nødmodus på armen. Denne nødmodusen ville fungert ved å benytte den samme funksjonen som detekterer kollisjoner, men med muligens en annen margin. Når systemet oppdager en mulig kollisjon, ville armen gå inn i en nødmodus hvor den stopper opp, justerer endepunktet og deretter fortsetter bevegelsen. Dersom det fremdeles er fare for kollisjon etter flere forsøk, ville armen trekke seg tilbake til utgangsposisjonen og utløse en melding om behov for assistanse i det aktuelle området.

Den foreslåtte endringen ville ha flere fordeler, inkludert økt sikkerhet og pålitelighet ved håndtering av mulige kollisjoner. Imidlertid ville det også være viktig å vurdere mulige utfordringer og begrensninger knyttet til implementeringen av denne løsningen, for eksempel ytelsesevne eller kompleksitet i systemet.

Samlet sett vil forhindring av kollisjoner være et viktig skritt for å forbedre systemets funksjonalitet og pålitelighet. Ved å implementere en nødmodus som beskrevet, vil vi kunne styrke systemets evne til å håndtere potensielle kollisjonssituasjoner på en effektiv og trygg måte.

7.7 HVL_fcDistanceFromPointToLine

For denne funksjonsblokken lagde vi originalt en versjon som fikk inn indeks til tabellen med alle punkter fra LiDAR, disse indeksene referere til punkter for start og sluttpunkt på linjen og et punkt for punktet som skal sjekkes. Vi innså at det ville vært mer hensiktsmessig og nødvendig å endre funksjonen til å ta inn en linje og ikke indeks til en tabell. På grunn av tidsbesparelse valgte vi å lage en ny kopi av funksjonen som får inn en linje. Dette gjorde vi for å slippe å endre koden fra tidligere som bruker funksjonen. Vi fikk da mulighet til å lage nye prosesser som trengte å beregne avstand fra punkt til en linje uten at de var avhengig av de originale målte punktene

7.8 Eksport av data for sammenligning

I oppgaven var det definert at vi skulle eksportere data fra ferdigtegnede linjer til Excel for å sammenligne data med originale punkter fra LiDAR-sensor. For enkelhets skyld utførte vi datasammenligningen i Geogebra. Dette gjorde vi grunnet Geogebra sine gode egenskaper til å visualisere koordinatsystem.

Valgte som sagt å sammenligne de originale punktene fra LiDAR-sensoren med krysningspunktene vi hadde funnet. Dette ble gjort fordi det var disse punktene som ble brukt til å visualisere linjene på det ferdige, detekterte måleobjektet på HMI-skjermen. Dersom vi hadde hatt mer tid, ville vi også ha inkludert punktene fra de originale linjene for å bedre kunne observere utviklingen gjennom hele prosessen med linjedeteksjon.

7.9 Egnethet LiDAR

For en slik oppgave hadde en LiDAR-sensor både sine fordeler og ulemper. En fordel er at LiDAR er en enkel sensor som ikke er avhengig av flere sensorer, man trenger kun selve LiDAR-sensoren. I tillegg vil remisjonsverdien kunne brukes til å markere kritiske punkt. For eksempel ved bruk av refleksteip kan man markere ønskede områder som LiDAR kan skille fra en generell vegg. LiDAR-sensoren kan samle inn data raskt, dermed vil man få mye data på kort tid. Generelt sett er dette bra, for man vil oppnå målinger så nær sanntid som mulig. Men i samspill med en PLS kan det være en utfordring med tanke på begrenset prosesseringskraft. Ellers er også LiDAR-sensoren til en viss grad sårbar for forstyrrelser.

8 Konklusjon

Gjennom utviklingen av vårt system har vi konvertert polare koordinater fra LiDAR-sensoren til kartesiske koordinater, som deretter har blitt anvendt for linjedeteksjon i måleobjektet. Etter vellykket deteksjon, har vi implementert en prosess for plassering av et forhåndsdefinert måleobjekt langs disse linjene, etterfulgt av grundig validering for å bekrefte korrekt posisjonering. Vårt system er også utviklet med evnen til å tilrettelegge for inspeksjon med en mekanisk arm uten kollisjon. Vi har prioritert håndtering av forstyrrelser, inkludert tilfeller der den mekaniske armen potensielt kan dekke synsfeltet til LiDAR-sensoren. Dette er oppnådd ved å opprettholde synligheten av et referansepunkt på måleobjektet, som gjør det mulig å følge objektets bevegelser, selv i tilfeller der LiDAR-sensoren har begrenset synsfelt. Videre har systemet blitt utviklet med funksjonaliteter for å detektere kollisjoner mellom den simulerte armen og systemets vegger.

Mens vi er tilfredse med våre prestasjoner, gjenstår det fortsatt noen områder som krever ytterligere arbeid. Spesielt fokuserer dette på å forbedre robustheten og håndteringen av forstyrrelser, som eksempelvis utfordrende værforhold. Etter samtale med intern veileder hos bedrift, ble vi enige om å sette kravet om toleranse mot værforhold som en opsjon i stedet for et krav (Vedlegg, Møtereferat 2024-04-12, s. 2). Fremtidig arbeid bør derfor prioritere disse aspektene for å sikre at systemet kan fungere optimalt under ulike forhold.

Bacheloroppgaven har vært en utfordrende, men verdifull læringsprosess for oss som har bidratt til å utvide vår kunnskap og erfaring innen utvikling av systemer basert på LiDAR-teknologi og PLS. På tross av mangelen på en klar problemstilling ved starten av prosjektet, har vi lykkes med å utvikle et system som kan håndtere data fra LiDAR-sensoren og PLS-en for å detektere og plassere et måleobjekt. Selv om enkelte deler av prosjektet kanskje ikke ble løst helt som planlagt, er vi tilfredse med resultatet og føler at vi har oppnådd de fleste målene som ble satt ved starten av oppgaven.

Referanser

[1] Proff.no, «Proff.no,» 2022. [Internett]. Available: https://www.proff.no/selskap/controlteam-as/nesttun/automasjon-og-utstyr/IG7KFL40ZDA. [Funnet 18 01 2024].

[2] «SNL,» [Internett]. Available: https://snl.no/lidar. [Funnet 26 01 2024].

[3] «NDLA,» [Internett]. Available: https://ndla.no/nb/subject:1:8c5a9fdd-4fa4-456b-9afe-34e7e776b4e7/topic:ce841519-de73-4349-870f-2240e5276bc0/resource:3eca6f64-0408-46bc-aa3c-827c7ead3352. [Funnet 26 01 2024].

[4] Siemens, «Siemens HMI,» [Internett]. Available: https://support.industry.siemens.com/cs/pd/1430611?pdti=pi&dl=en&lc=en-EG. [Funnet 23 04 2024].

[5] MATLAB, [Internett]. Available: https://www.mathworks.com/products/MATLAB.html. [Funnet 05 04 2024].

[6] Siemens, 26 04 2024. [Internett]. Available: https://www.siemens.com/global/en/products/automation/industry-software/automation-software/TIA-portal.html.

[7] Pepperl+Fuchs, [Internett]. Available: https://www.pepperl-fuchs.com/global/en/classid_53.htm?view=productdetails&prodid=86558. [Funnet 05 04 2024].

[8] Siemens Industry Mall, «Webområde for Siemens Industry Mall,» 01 04 2024. [Internett]. Available: https://mall.industry.siemens.com/mall/en/WW/Catalog/Products/10239949?activeTab=productinformation&regionUrl=WW. [Funnet 26 04 2024].

[9] Siemens Industry Mall, «Siemens Industry Mall,» 01 04 2024. [Internett]. Available: https://mall.industry.siemens.com/mall/en/WW/Catalog/Products/10360595?activeTab=ProductInformation. [Funnet 26 04 2024].

[10] «IE61131-3,» [Internett]. Available: https://plcopen.org/iec-61131-3. [Funnet 29 04 2024].

[11] «Wikipedia,» 06 06 2023. [Internett]. Available: https://en.wikipedia.org/wiki/Distance_from_a_point_to_a_line. [Funnet 27 02 2024].

[12] M. Milford, «Math-mate,» [Internett]. Available: https://www.math-mate.com/chapter39_2.shtml. [Funnet 13 05 2024].

[13] «Jula,» [Internett]. Available: https://www.jula.no/catalog/bygg-og-maling/oppbevaring/oppbevaringshyller-skuffer/oppbevaringsskuffer-kurver/trekasse-009042/. [Funnet 19 04 2024].

[14] «Geogebra,» [Internett]. Available: https://www.geogebra.org/about. [Funnet 19 04 2024].

[15] Panduro, «Panduro,» [Internett]. Available: https://panduro.com/nb-no/products/papir-stempler/dekorpapir/papirsett/sett-a4-ark-80-g-46-stk-365421. [Funnet 26 04 2024].

[16] Jula, «Jula,» [Internett]. Available: https://www.jula.no/catalog/bygg-og-maling/maling-og-fugemasse/teip-og-maskering/spesialteip/refleksteip-021488/. [Funnet 26 04 2024].

Figur Liste

Tabell Liste

Formel Liste

Appendiks A Forkortelser og ordforklaringer

CPU	Central Processing Unit
HMI	Human Machine Interface
LiDAR	Light Detection And Ranging
PLS	Programmerbar logisk styring
SW	Programvare (Software)
HW	Maskinvare (Hardware)

Appendiks B Vedlegg beskrivelse

Til denne rapporten er det vedlagt en komprimert-mappe som inneholder vedleggene våre.

- TIA-Portal prosjektfilen.
- Kode skrevet ut i PDF-format, slik at kode kan leses uten TIA-Portal programmet.
- Fremdriftsplan.
- LiDAR konfigurasjonsfilen (pactware).
- Midtveispresentasjonen.
- Møtereferat.
- Timeliste.
- Videoer fra HMI-panelet.