UNIVERSITETET I AGDER

IMPLEMENTERING AV MASKINSYN FOR MONTERING AV FASADEPLATER

Remi Bendiksen Askeland

Veileder
Professor Geir Hovland

Masteroppgaven er gjennomført som ledd i utdanningen ved Universitetet i Agder og er godkjent som del av denne utdanningen. Denne godkjenningen innebærer ikke at universitetet innestår for de metoder som er anvendt og de konklusjoner som er trukket.

Universitetet i Agder, 2018
Fakultet for Teknologi og Realfag
Institutt for Ingeniørvitenskap

Forord

Denne avhandlingen er et produkt av kurset MAS500, Masteroppgave i mekatronikk ved Universitetet i Agder (UiA). Kurset går over et semester og er kreditert 30 studiepoeng. Prosjektet ser på implementering av maskinsyn for montering av fasadeplater. Motivasjonen bak dette prosjektet er et samarbeid mellom Universitetet i Agder og MacGregor Norway AS.

Jeg ønsker å takke alle som har bidratt med sin ekspertise under prosjektperioden. Første takk går til professor Geir Hovland, for jevnlig veiledning og hjelp under arbeidet med avhandlingen. Vil takke lab assistent Erind Ujkani for sin ekspertise innen C++ og OpenCV under perioden. Takk til Beckhoff Norge som har vært til stor hjelp med sin gode kundesupport. Ønsker også å takke MacGregor og UiA for tildeling av en spennende og utfordrende oppgave samt for samarbeidet under perioden. Til slutt ønsker jeg å takke Phd. Kandidat Sondre Sanden Tørdal som var til stor hjelp under avhandlingsperioden med sin ekspertise innen robotikk og maskinsyn.

Remi B. Askeland

26.01.2018
Universitetet i Agder, Grimstad

Sammendrag

Effektivisering av byggebransjen har fått mye oppmerksomhet de siste årene. MacGregor i samarbeid med Universitetet i Agder vil bruke kran- og løfteteknologi til å effektivisere enkelte byggeprosesser. Fasadeplater brukes mye på grunn av enkel installasjon og krever minimalt med vedlikehold. Prosessen assosiert med installasjon har et automatiserings potensiale.

Denne avhandlingen ser på implementering av maskinsyn for montering av fasadeplater. Monteringen blir utført med en ABB IRB6600 industrirobot og et vakuumverktøy. Maskinsynet er basert på OpenCV biblioteket og C++. Hovedkomponentene i kontrollsystemet er en Beckhoff CX2040 Embedded-PC, Basler acA2500 kamera og Techspec 4mm linse.

Maskinsynet er bygget opp med OpenCV algoritmer som lokaliserer og estimerer fasadeplatens lokasjon. Maskinsynet brukes også til å plassere fasadeplaten ut fra allerede monterte plater. Utfordringene i avhandlingen var kalibrering av linse og kamera for å fjerne geometrisk forvrenging forårsaket av linsen. Gjentatte tester viser god ytelse i senter av kamera og mindre tilfredsstillende resultater andre steder i kameraets synsfelt.

Beckhoff CX2040 brukes som hovedkontroller i systemet og styrer inngang- og utgangssignaler, systemlogikk, maskinsyn og HMI. Kontrolleren viser seg å takle enkle maskinsyn-oppgaver og fungerte utmerket under avhandlingens periode.

Summary

The construction industry has seen a surge in efficiency and automation solutions in recent years. MacGregor, in cooperation with the University of Agder want to use crane and lifting technology to streamline construction processes. Facade panels are widely used due to ease of installation and require minimal maintenance. The process associated with installation has an automation potential.

This thesis looks at the implementation of machine vision for installation of facade panels. Installation is done using a ABB IRB6600 industrial robot and a vacuum tool. The machine vision is based on OpenCV and C++. The main components of the control system are a Beckhoff CX2040 Embedded-PC, Basler acA2500 camera and a Techspec 4mm lens.

The machine vision is developed using OpenCV algorithms that locate and estimate the position of the facade panels. The machine vision is also used to install the facade panels according other objects on the wall. The challenges were the calibration of lens and camera to remove geometric distortion. Repeated testing shows good performance in the center of the camera and less sufficient results elsewhere in the camera's field of view.

The Beckhoff CX2040 is used as the main controller in the system. It controls input and output signals, system logic, machine vision and HMI. The controller turns out to handle simple machine vision tasks and worked excellent during the dissertation period.

Innholdsfortegnelse

Tabeller

Figurer

Forkortelser

Forkortelse	Forklaring
UiA	Universitetet i Agder
MS	Maskinsyn
PLS	Programbar logisk styring
I/O	Innganger/Utganger (engelsk: input/output)
MMG	Menneske-maskin grensesnitt
HMI	Human maskin interface
CCD	Charge-coupled device
CMOS	Complementary metal oxide semiconductor
Mp	Megapiksler
LPF	Lavpassfilter
HPF	Høypassfilter
PnP	Perspektiv-til-Punkt
ML	Maskinlæring
IP	Internettprotokoll
ADS	Automation device specification
VS	Verktøy senter
POU	Programorganiseringsenheter
FB	Funksjons blokk
PGR	Program
STL	Structured text list
RT	Sanntid (engels: real time)
N-RT	Ikke sanntid (engelsk: Non-Real time)

1 Introduksjon

MacGregor Norge vil bruke kran- og løfteteknologi fra offshore industrien for å automatisere og effektivisere byggeprosesser [1]. I samarbeid med Universitetet i Agder ser MacGregor på mulighetene for å utvikle en robotarm til byggebransjen. Denne er i første omgang tenkt for montering av fasadeplater på utsiden av bygg.

Prosessen med å montere fasadeplater er i dag en tidkrevende og arbeidsintensiv prosess som krever mye utsyr assosiert med montering og for å sikre involvert personell. Prosessen er repetitiv og har en del variabler som gjør det mulig å utforske mulighetene for å automatisere prosessen.

Oppgaven med monterering av fasadeplater er krevende for en robot. Platene skal plasseres ut i fra andre plater, vinduer, dører, kanter på veggene eller lignende. Monteringen må være nøyaktig for å få et symmetrisk resultat. For å kunne utføre oppgaven med å plassere fasadeplater med robot trengs sensorer for lokalisering, måling og kvalitetskontroll. En teknologi som kan utføre alle disse oppgavene er maskinsyn(MS)

Maskinsyn er et fagfelt som er blitt veldig populært de senere årene da kamerateknologien blir bedre og datakraften nødvendig blir billigere. Industrien bruker maskinsyn-teknologi til blant annet å sortere frukt og kvalitetskontrollere maskindeler. Teknologien er også blitt populær kommersielt og brukes blant annet til å låse opp I-Phone med ansiktsgjenkjenning eller innen selvkjørende biler. Utviklingen av maskinsyn skjer fort på grunn av store nettbaserte samfunn med åpen kildekode. OpenCV er utviklet av slike nettsamfunn og er et bibliotek med algoritmer for maskinsyn.

Hovedformålet med denne avhandlingen er å bygge opp et robotsystem som bruker maskinsyn. OpenCV skal implementeres og testes ut i denne avhandlingen. Robotcellen er bygget opp for å simulere prosessen med og montere fasadeplater. Eksperimentelle tester gir basis for resultatene presentert i denne avhandlingen.

1.1 Oppgavebeskrivelse

Denne avhandlingen vil se på implementering av maskinsyn og OpenCV. Avhandlingen vil også se på repeterbarhet ved bruk av maskinsyn. Det ble spesifisert at Beckhoff CX2040 brukes som hovedkontroller i systemet. Roboten tildelt for dette prosjektet er en ABB IRB6000 1.75/2.55. Systemet skal simulere prosessen med å montere fasadeplater.

Underoppgaver:

- Finne og implementere et kamera.
- Sette opp Beckhoff CX2040 for kontroll av hovedfunksjoner, inngangs- og utgangssignaler (I/O) og kommunikasjon.
- Implementere OpenCV for objektdetektering og kantdetektering samt måle behandlingstid og ressursforbruk.
- Bygge en vegg for å teste plasserings nøyaktighet.
- Lage menneske-maskin grensesnitt som kjører på Beckhoff CX2040

Se vedlegg A for oppgavebeskrivelse gitt av UiA og MacGregor.

1.2 Avhandlingens oppbygging

Denne avhandlingen er bygget opp av 5 hovedkapitler, referanser og vedlegg. Dette kapitelet introduserte avhandlingen med oppgavebeskrivelsen. Kapittel 2 går gjennom den relevante teorien for å finne en god løsning. Videre i kapittel 3 vil metodene og teknologien brukt for å løse oppgave forklares. Oppnådde resultater og systemets ytelse er beskrevet i kapittel 4. Kapittel 5 diskuterer systemets ytelse og hvordan det kan forbedres. Konklusjonen er presentert i kapittel 6. Referansene brukt i avhandlingen finnes i kapittel 7,og nødvendige vedlegg i kapittel 8

2 Teori

2.1 Maskinsyn

En robot er en målbevisst maskin som kan føle, planlegge og utføre, dette er definisjonen av en robot beskrevet i *Robotics, Vision and Control* [2]. I dette kapittelet blir det sett på teorien bak kamera teknologi og maskinsyn. Dette danner basen for hvordan maskinsyn kan brukes til å hjelpe en robot å navigere.

2.1.1 Kamera

Maskinsyn starter med bildeinnsamling. Et kamera inneholder en sensor som registrerer lys som passerer gjennom en optisk linse. Nærmere bestemt måler normale kamera energi eller elektromagnetiske bølger som reflekteres fra omgivelsene. Den optiske linsen begrenser hvor mye lys som kan treffe sensoren som gir et lesbart bilde.

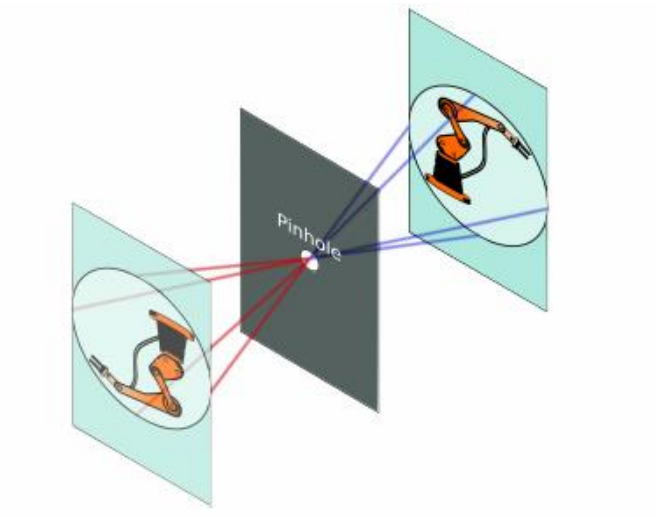
Det er to hovedprinsipper som brukes for å konvertere lysbølger til digitale verdier: CCD (engelsk: charge-coupled device) og CMOS (engelsk: complementary metal oxide semiconductor). Sensorer med CCD måler spenningen akkumulert under sensorens eksponeringstid. Dette spenningssignalet blir forsterket og konvertert til digital verdi etter eksponeringstiden er over. Sensorer med CMOS teknologi måler og forsterker spenningen direkte i hver piksel, denne type sensor er den mest vanlige. Sensoren består av mange piksler, der en piksel registrerer mengden lys eller energi som den blir utsatt for. Eksponeringstiden bestemmer hvor mye energi pikselen blir utsatt for. Dersom eksponeringstid er for høy blir bildet overeksponert og for lyst, for liten eksponeringstid og bilde blir mørkt og under eksponert. De fleste kamera måler mengden innkommende lys for å få korrekt eksponeringstid. Mengden piksler bestemmer oppløsningen og detaljnivået i bildet. Dette betyr at objekter i bilde kan beskrives av flere piksler jo høyere piksel antallet blir. Oppløsning beskrives med antall mega piksler (Mp). 5 Mp betyr at sensoren har fem millioner individuelle piksler.

Kamera og bildeelektronikk spiller en viktig rolle i ytelsen i maskinsyn. Riktig integrering av alle komponenter som kamera, programvare og kabler er viktig for systemets stabilitet og ytelse. For å velge riktig kamera til applikasjonen må noen punkter vurderes:

- Oppløsning.
- Grensesnitt for dataoverføring (GigE, USB, og så videre)
- Grensesnitt for strøm.
- Enkelfarge- eller fargesensor
- Antall bilder per sekund
- Fysisk størrelse på kamera

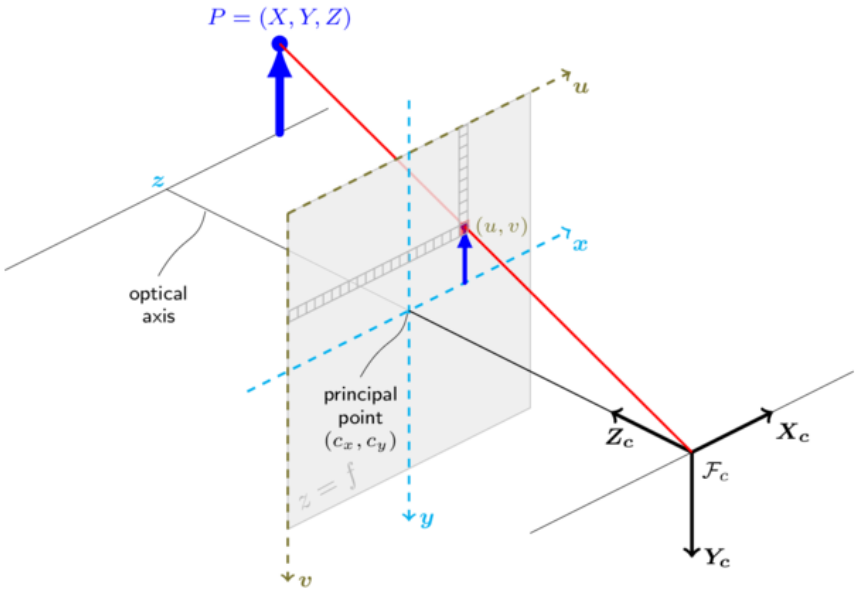
2.1.2 Kameramodellen

Hullkameramodellen blir brukt for å beskrive det matematiske forholdet mellom et punkt i det tredimensjonale rom og punktet projisert i bildeplanet. Denne modellen blir brukt av OpenCV [3] og andre [2]. Modellen beskriver hvordan lys reflekteres av et objekt og passerer gjennom et veldig lite hull i kameraet. Resultatet er 2D-projeksjon av et 3D-punkt eller objektet på bildeplanet. Hullkamera modellen er illustrert i Figur 2.1



Figur 2.1 Hullkameramodellen

Når et 3D-punkt projiseres til et 2D-plan forsvinner en dimensjon, nemlig dybden. Dette kan observeres i Figur 2.2. Punktet $P(X, Y, Z)$ blir projisert over til bildeplanet og representeres av $p(u, v)$.



Figur 2.2 Matematisk hullkameramodell [3]

Hullkameramodellen er idealistisk og tar ikke hensyn til effekter som geometrisk forvrenging og sløring. Disse effektene kan motvirkes ved å kalibrere kamera for å skape et bilde som bedre representerer virkeligheten.

2.1.3 Kamerakalibreing

Når et kamera skal brukes i maskinsyn for posisjonsestimering må det kalibreres for å ta hensyn til at sensor og linse ikke er perfekt. Kameraet må kalibreres for å finne de interne parametere, beskrevet av K. En sensor med 5 Mp og en bredde på 2592 piksler og høyde på 1944 piksler skal idealt ha sitt senter i 1296, 972. Sensoren er derimot ikke helt perfekt og kan hypotetisk sett ha sitt senter i 1295,973. Senterpunktet er beskrevet i piksler av u_0 og v_0. De interne parameterne beskrevet i ligning 1 tar også hensyn til fokallengde f i mm og bredde p_w og høyde p_h av hver piksel.

$$K = \begin{bmatrix} \frac{f}{pw} & 0 & u_0 \\ 0 & \frac{f}{p_h} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

Kameraets rotasjon $R(3x3)$ og translasjon $T(3x1)$ kombineres til T_c^0 vist i ligning 2.
T_c^0 er en homogen transformasjonsmatrise og er en generisk og kompakt måte å beskrive forholdet mellom to koordinatsystemer, dette forklares bedre i 2.5

$$T_c^0 = \begin{bmatrix} R_{3x3} & T_{3x1} \\ 0_{1x3} & 1 \end{bmatrix} \tag{2}$$

Punkt $\tilde{p}(p, 1)$ i ligning 3 beskriver verdens punkt P i homogene piksel koordinater. De indre og ytre parameterne kombineres til kameramatrisen C. $\tilde{P}(P, 1)$ er den homogene beskrivelsen av punktet P

$$\begin{aligned} \tilde{p} &= \begin{bmatrix} \frac{f}{pw} & 0 & u_0 \\ 0 & \frac{f}{p_h} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} T_c^0 \tilde{P} \\ &= K P_0 T_c^0 \tilde{P} \\ &= C\tilde{P} \end{aligned} \tag{3}$$

Under kamerakalibrering kan også forvrenging i linsen modelleres. Forskjellige typer avvik dannes av ujevnheter i linsen og sensorens vinkel i forhold til linsen. Disse avvikene er kromatisk avvik, sfærisk avvik, fokus avvik og geometrisk forvrenging. Geometrisk forvrenging er den mest problematiske effekten [2] når kamera skal brukes til posisjonsestimering. Geometrisk forvrenging kan modelleres og beskrives av ligning 4 der δ_u og δ_v er forskyvning i henholdsvis u og v retning

$$\begin{aligned} u^d &= u + \delta_u \\ v^d &= v + \delta_v \end{aligned} \tag{4}$$

Geometrisk forvrenging består av to komponenter, radiell og tangentiell forvrenging, disse modelleres ved å bruke ligning 5. Radiell fører til at punkter forflyttes langs radielle linjer fra senterpunktet. Tangentiell forårsaker punktene til å forflytte seg rettvinklet i forhold til de radielle linjene. Radiell forvrengning er den mest dominerende av de geometriske forvrengningene.

$$\begin{pmatrix}\delta_u\\ \delta_v\end{pmatrix} = \underbrace{\begin{pmatrix}u(k_1r^2+k_2r^4+k_3r^6+\cdots)\\ v(k_1r^2+k_2r^4+k_3r^6+\cdots)\end{pmatrix}}_{\text{Radiell}} + \underbrace{\begin{pmatrix}2p_1uv+p_2(r^2+2u^2\\ p1(r^2+2v^2)+2p_1uv\end{pmatrix}}_{\text{Tangentiell}} \quad (5)$$

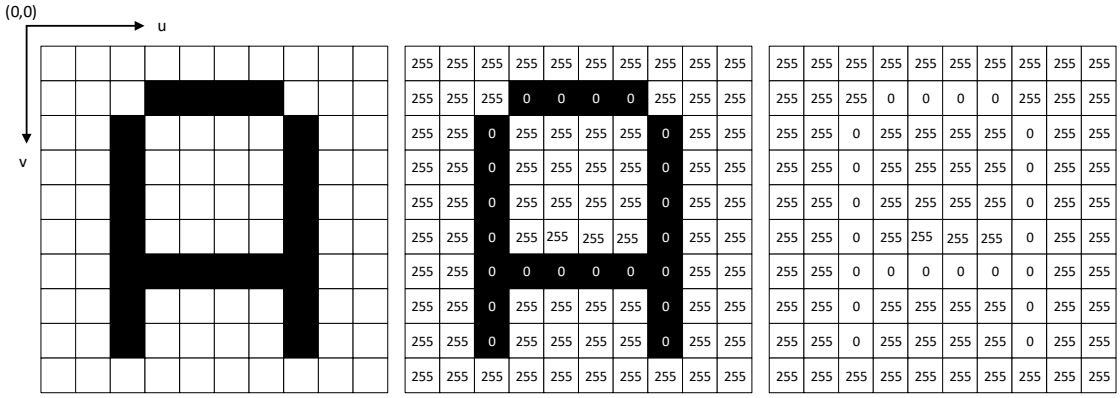
Kamera kalibrering finne alle parameterne beskrevet i dette kapittelet. Det finnes en rekke metoder og algoritmer for å estimere indre- og ytre parametere samt forvrenging. MATLAB og OpenCV har innebygde algoritmer for kamera kalibrering. MATLABs kalibrering er basert på "*A Flexible New Technique for Camera Calibration"* [4] og "*A Four-step Camera Calibration Procedure with Implicit Image Correction"* [5].

Kalibreringsprosedyren gjennomføres på følgende metode [4]:

- Bruk et mønster av kjent dimensjon festet til et plant underlag.
- Ta bilder av mønsteret fra forskjellige vinkler og avstander.
- Detekter kjente punkter i bilde.
- Estimer parametere og forvrenging.

2.1.4 Bildebehandling

Et bilde er ofte representert av 8bit (1byte) for hver piksel. Dette gir 256 mulige verdier. 0 representerer svart mens 255 representerer hvit. Alt imellom er forskjellige nyanser av grå. Figur 2.3 viser hvordan et bilde er representert med menneske til venstre og maskin perspektiv til høyre. Figur 2.3 viser også den mest brukte konvensjonen når det kommer til koordinatsystem i bildeplanet, nullpunktet ligger oppe i venstre hjørne av bilde.

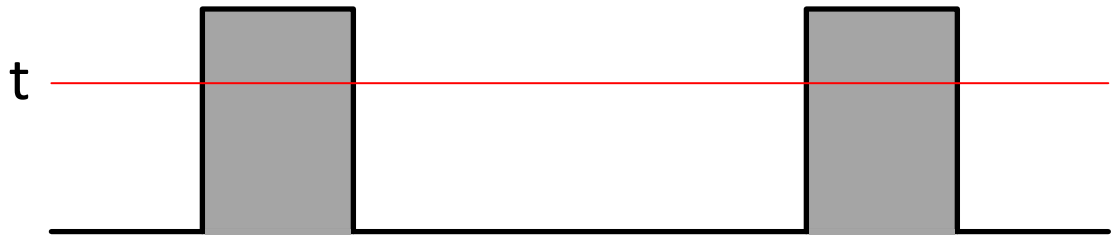


Figur 2.3 Piksel representasjon

For å representere farger brukes flere sensorer. Lyset blir splittet opp gjennom speil og optiske filtre. Normalen er å bruke de 3 primærfargene rød, grønn og blå for å representere forskjellige farger. Hver av fargene representeres av 8 bit som gir $256^3 = 16{,}777{,}214$ mulige verdier en piksel kan ha. Et fargebilde er mer komplisert matematisk enn et gråtone bilde.

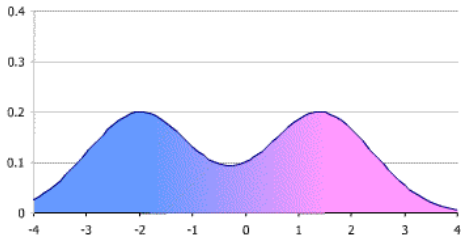
Det er mange strategier for å manipulere et bilde for å få ønsket resultat, to av disse er punktoperasjoner og nabolagsoperasjoner. Punktoperasjoner utfører en operasjon på en enkel piksel $f(u,v)$ som fører til endringen $g(u,v)$. Et eksempel på er å legge til en verdi til en piksel for å gjøre den lysere eller mørkere. En vital punktoperasjon er grenseverdi (engelsk: thresholding). Ved binær grenseverdi operasjon settes en grenseverdi, t, vist i Figur 2.4. Pikselverdiene blir da omgjort til 0 eller 255 uti fra om den er over eller under grenseverdien. Formel for binær grenseverdi operasjoner beskrevet i OpenCV [6] er gitt i ligning 6

$$g(u,v)\begin{cases}255\ dersom\ f(u,v) > t \\ 0\ dersom\ f(u,v) \leq t\end{cases} \tag{6}$$



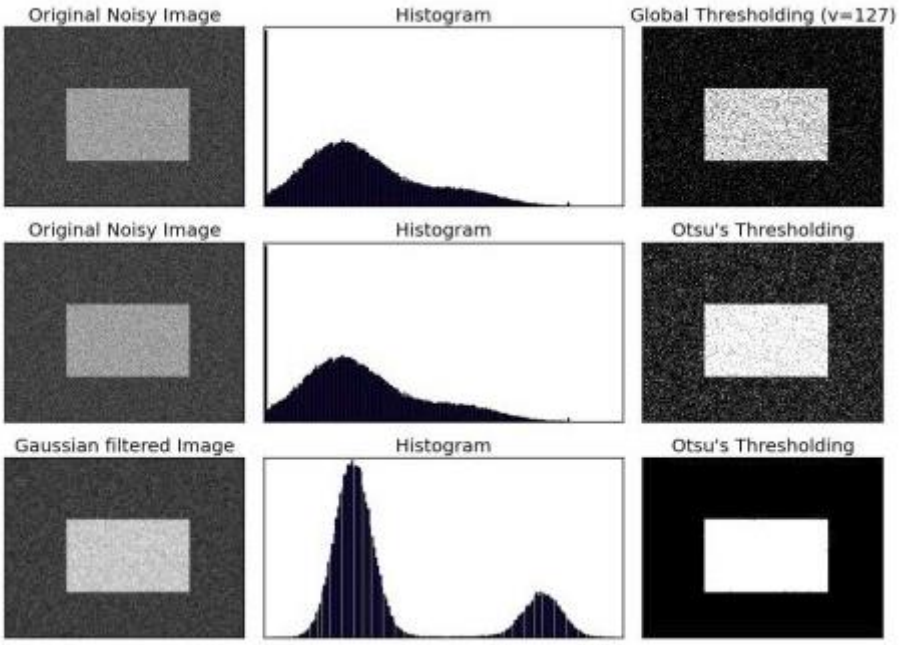
Figur 2.4 Grenseverdi operasjon

Otsu metoden [7] er en grenseverdi metode som velger passende grenseverdi selv. Metoden antar at bilde inneholder to klasser med piksler, bakgrunn og forgrunn(objekt), og at disse følger en bimodal fordeling. En bimodal fordeling er en kontinuerlig sannsynlighetsfordeling med to ulike moduser. Disse opptrer som distinkte topper eller lokale maksimum som vist Figur 2.5.



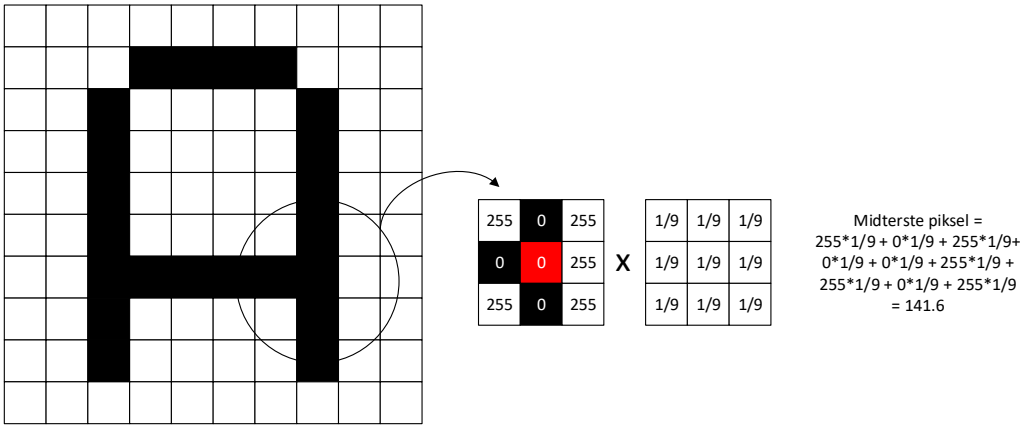
Figur 2.5 Bimodal fordeling

Otsu metoden finner den optimale grenseverdien slik at de to klassene blir separert så mye som mulig og den indre variansen i hver klasse minimeres, se Figur 2.6. Metoden har sine begrensninger og når objektet i bilde er lite i forhold til bakgrunnen kan dette føre til dårlig bimodal fordeling. En annen begrensing er dersom bilde inneholder mye tilført støy som fører til at dalene i histogrammet ikke er bratt nok kan dette føre til ukorrekt grenseverdi



Figur 2.6 Otsu grenseverdi metode

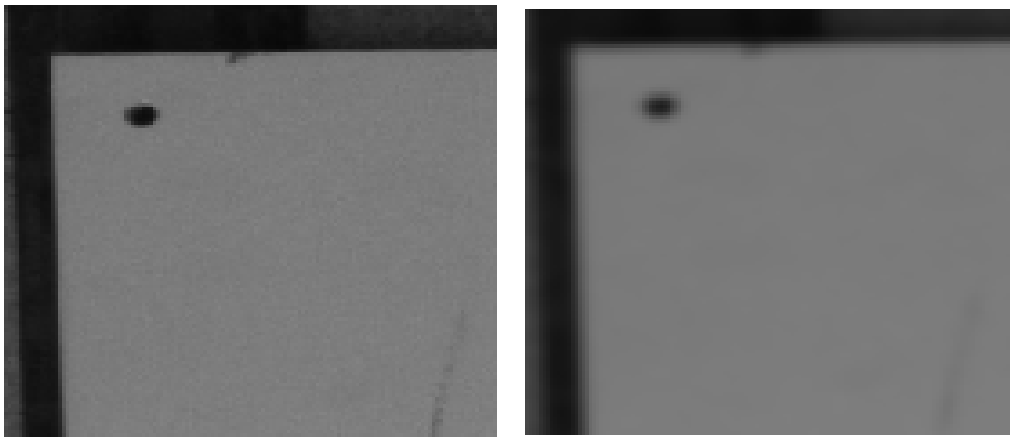
Nabolagsoperasjoner utfører operasjoner på en piksel basert på pikslene rundt. Nabolagsoperasjoner kan virke som lavpassfilter(LPF) eller høypassfiltre(HPF). Slør filter (engelsk: blur) er et lavpassfilter som reduserer støy. Filteret tar gjennomsnittet av pikslene rundt den aktuelle pikselen og erstatter med gjennomsnittsverdien. Hvor mange piksler rundt senter som brukes kalles en kjerne. Figur 2.7 viser en enkel 3x3 kjerne og resulterende verdi på senterpikselen.



Figur 2.7 Nabolagsoperasjon

Normalfordelt LPF eller Gaussisk LPF bruker et filter med varierende vekting av pikslene rundt senter. Filteret fjerner høyfrekvente komponenter og støy. Ligning 7 viser hvordan verdien for hver piksel er kalkulert. u og v er koordinatene og σ er standard avviket av den gaussiske fordelingen. σ bestemmer hvor mye bilde blir glattet. Figur 2.8 viser til venstre et platehjørne uten filter og til høyre et Gaussisk LPF tilført samme bilde

$$G(u,v) = \frac{1}{2\pi\sigma^2} e^{\frac{u^2+v^2}{2\sigma^2}} \tag{7}$$



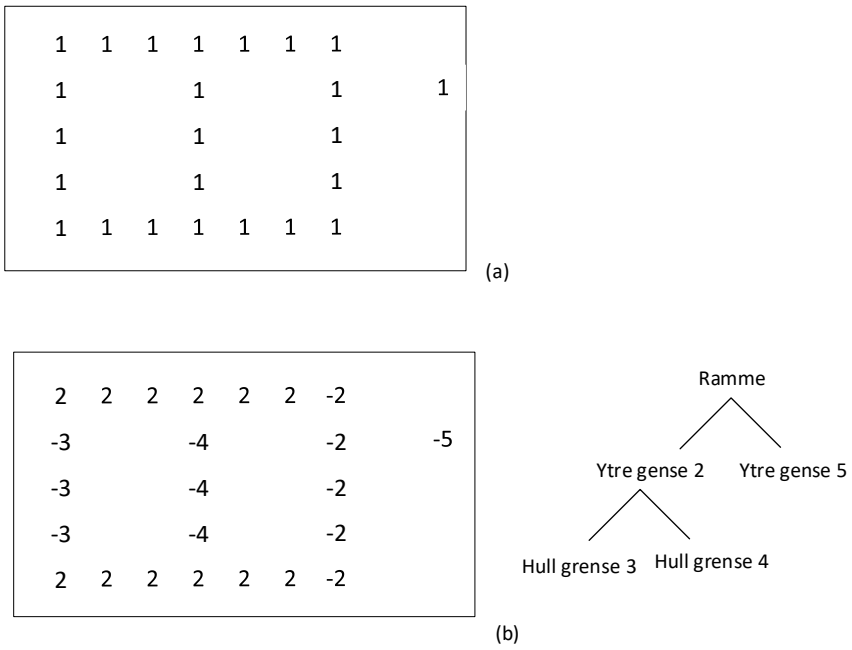
Figur 2.8 Gaussisk filter

2.1.5 Konturer

En vital del av maskinsyn er å finne konturer og former. Analyse av binære bilder og topologisk struktur har gitt forskjellige algoritmer som løser dette. En slik algoritme er «Suzuki & Abe» grense følging (engelsk: border following) [8]. Og finne en grense eller kontur av et objekt blir gjort ved å følge sammenhengende piksler med samme intensitet eller farge. Denne metoden krever et binært bilde.

Algoritmen antar at bildet består av 0-piksler som beskriver bakgrunn og hull, og 1-piksler som beskriver sammenhengende konturer. Konturalgoritmen skanner bildet f_{ij} linje for linje og pikslene blir merket (i, j), hvor i beskriver rad og j beskriver kolonne hvor pikselen befinner seg i. Hvis en piksel blir funnet som tilfredsstiller kravene for begynnelse på en grense blir den gitt et sekvensielt nummer og navnet NBD. Dette er ikke en forkortelse men er navnet gitt til piksler som er den nyeste grensen funnet. Når algoritmen finner slutt på en grense eller et hull blir pikselen notert LNBD som beskriver den sist funne grensen.

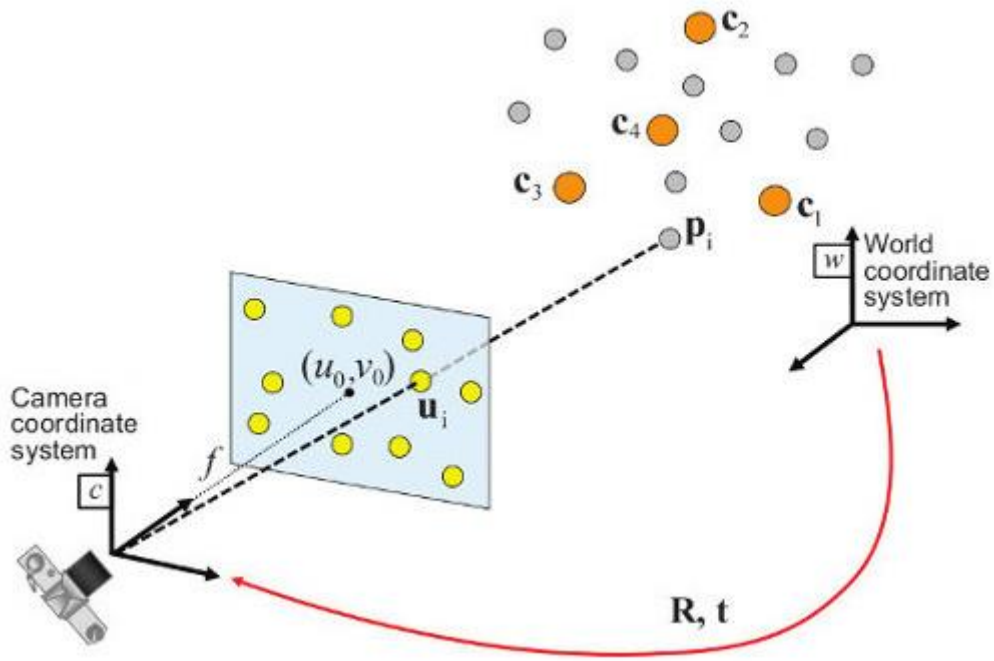
Figur 2.9a er et bilde der 1-pikslene er sammenhengende konturer og alt som ikke er 1-piksler er 0-piksler. Algoritmen skanner bilde, finner starten og slutten på konturene og hull. Figur 2.9b viser hvordan algoritmen har merket pikslene ut i fra om det er kontur start, slutt eller hull.



Figur 2.9 Illustrasjon av konturalgoritme

2.1.6 Posisjonsestimering

Kameraets posisjon kan beskrives av 6-frihetsgrader. Disser består av rotasjon om tre akser og 3D translasjon i forhold til et kjent koordinatsystem, dette forklares mer i et senere kapittel. Kameraets posisjon kan estimeres dersom et n-antall 3D-punkter i verden og deres korresponderende 2D-punkter i bildeplanet er kjent. Dette er kjent som et perspektiv-til-punkt (PnP) problem [9]. Dette problemet stammer fra kamera kalibrering men har mange andre bruksområder innen maskinsyn som inkluderer 3D posisjonsestimering.



Figur 2.10 Posisjonsestimering [10]

Gitt et sett med 3D-punkter p_i (X, Y, Z) utrykket i referanse koordinatsystem og deres 2D projiserte punkter u_i (u, v) kan kameras relative posisjon R og T finnes ved å løse ligning 8. Dette er illustrert i Figur 2.10 Posisjonsestimering . For å estimere kameraets posisjon må de indre kamera parameterne være kjent. Forskjellige metoder kan brukes for å estimere den relative posisjonen. "Levenberg–Marquardt" algoritmer [11] finner en posisjon som minimerer reprojeksjons-feilene, dette gjøres ved å summere de kvadratiske avstandene mellom de observerte punktene u_i og objekt punktene p_i.

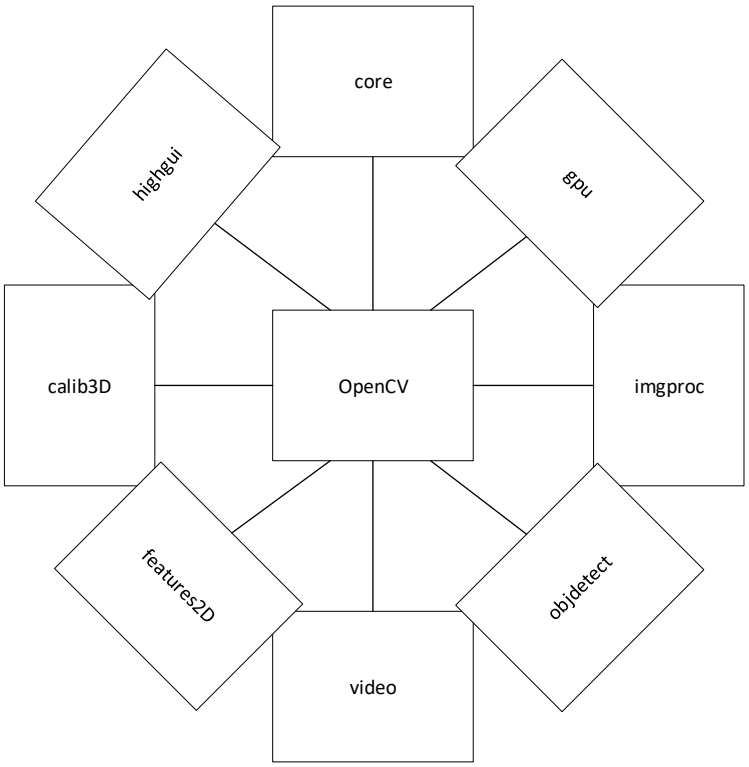
$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R_{11} & R_{12} & R_{13} & T_1 \\ R_{21} & R_{22} & R_{23} & T_2 \\ R_{31} & R_{32} & R_{33} & T_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \quad (8)$$

2.1.7 OpenCV

Open Source Computer Vision Library (OpenCV) er åpen kilde maskinsyn og maskinlærings programvarebibliotek. OpenCV er bygget for å gi en felles infrastruktur for maskinsyn utvikling. Biblioteket inneholder algoritmer, dokumentasjon og eksempler og dekker alt fra klassiske metoder til nymoderne algoritmer. Bibliotekene er åpen kildekode og lastes ned fra "OpenCV.org" [12]. OpenCV er lisensiert under 3-klausul av BSD [13] som gjør det enkelt for bedrifter å bruke eller modifisere kode. Store selskaper som Google, Toyota og Microsoft har tatt i bruk OpenCV. Det er også stor begeistring innen oppstartsmiljøet for bruk av biblioteket.

OpenCV har støtte for utvikling innen høynivå språk som C, C++, Python, Java og MATLAB. Biblioteket er originalt skrevet i C++ men har grensesnitt som støtter disse andre språkene. OpenCV har også støtte for bruk på de største plattformene som Windows, Linux, Android og Mac OS. Biblioteket lener seg mer mot sanntid bilde og video applikasjoner. Derfor er det i dag aktiv utvikling av grensesnitt mellom OpenCV, CUDA og OpenGL. Dette er for å senke prosesseringstiden ved bruk av GPU framfor CPU.

OpenCV har et utvikler- og brukersamfunn på ca 47000 registrerte brukere og er blitt lastet ned 14 millioner ganger. Dette gjør at det er mye dokumentasjon angående funksjoner og bruk av OpenCV. Dette store nettsamfunnet gjør det enkelt å få svar på spørsmål angående bruk, funksjoner og problemer. Det er heller ikke mangel på eksempler for bruk av OpenCV. Forum som Stackoverflow [14] og kode delings sider som Github [15] er gode ressurser. OpenCV blir løpende oppdatert med nytt innhold.



Figur 2.11 OpenCV struktur

OpenCV har en modulær struktur og er i hovedtrekk bygget opp som vist i Figur 2.11. En kort beskrivelse av modulene er gitt i Tabell 2.1.

Modul	Egenskap
Core	En kompakt modul som inneholder de grunnleggende data strukturene, som multidimensjonal matriser, `cv::Mat`, brukt av alle andre modulene
Improc	Bildeprosesseringsmodul som inkluderer lineær og ikke-lineære filtre, geometriske transformasjoner, fargespekter konvertering, histogram og lignende.
Video	Video analyse modul som inneholder bevegelses estimering, bakgrunns subtraksjon og objekt sporings algoritmer.
Calib3d	Modul som inneholder grunnleggende algoritmer for flere synsvinkler, enkelkamera kalibrering, stereokamera kalibrering, posisjonsestimering og 3D-rekonstuksjon
Features2d	Inneholder algoritmer for estimering av framtredende egenskaper og egenskaps gjenkjenning.
Objdetect	Modul med algoritmer for gjenkjenning av predefinerte objekter som mennesker, biler, katter og lignende.
Highgui	Et brukervennlig grensesnitt for video opptak, bilde og video kodeker
Gpu	GPU – algoritmer som kan brukes sammen med andre OpenCV moduler

Tabell 2.1 OpenCV moduler

2.2 Kontrollsystem

I dette kapittelet beskrives grunnleggende teori rundt industriell IT og kontrollsystemer. Teorien her danner grunnlaget for metodene og resultatene senere i avhandlingen.

2.2.1 Programmerbar logisk styring

En programmerbar logisk styring(PLS) er en robust datamaskin som er bygget for automatisering og industrielle applikasjoner. PLS brukes i produksjonslinjer, robotikk og andre aktiviteter som krever høy pålitelighet, enkel implementasjon og diagnose. Det finnes et stort utvalg av PLSer, fra små enkle med integrerte innganger og utganger til kraftige modulbaserte modeller. Leverandører av PLS og tilhørende utstyr er Beckhoff, Siemens, Allen Bradley og Omron for å nevne noen.

Hovedforskjellen mellom vanlige datamaskiner og PLS er muligheten til å koble til inngangs og utgangs signaler (I/O). Signalene kan enten være digital eller analoge. I/O håndteringen gjør det mulig å lese av sensorer og kontrollere aktuatorer.

PLSen opererer i en repeterende syklus. Selv om en prosess oppnår stasjonær tilstand kreves kontinuerlig overvåkning dersom forstyrrelser skulle oppstå. PLS utfører de samme 4 operasjonene under en normal syklus beskrevet i *Programmerbar Logisk Styring* [16]. Disse er intern prosessering, lese innganger, programutførelse og oppdatere utganger

En PLS programmeres ved å bruke programvare utformet for oppgaven på en vanlig PC. PLS programmet blir så overført til PLS gjennom USB, Ethernet eller RS kabler. PLS programmet blir lagret på PLS. PLS blir programmert under IEC 61131-3 standarden. Denne definerer blant annet fem språk for programmering:

- Funksjonsblokk diagram (FBD),
- Ladder diagram (LD)
- Strukturert tekst (ST)
- Instruksjonslister (IL)
- Sekvensielle funksjonskart (SFC).

De fundamentale konseptene for PLS programmering er lik for alle fabrikantene som bruker IEC 61131-3 standeren, men adressering av I/O signaler og organisering av minne er ofte forskjellig. Dette betyr at PLS programmer sjeldent kan byttes mellom PLS av forskjellig leverandører.

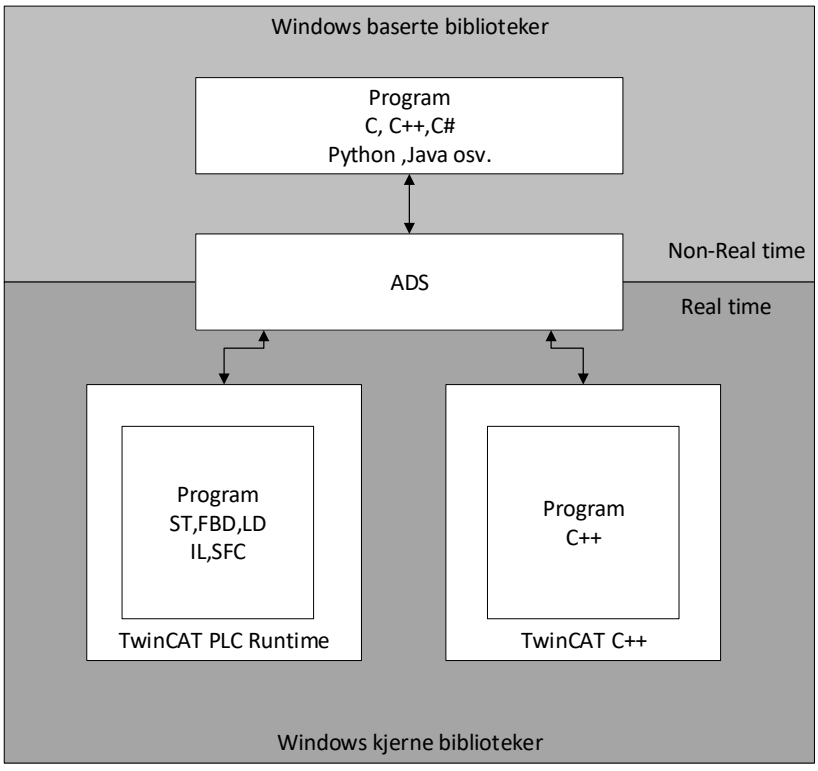
Enkelte leverandører har kombinert PCens allsidighet og PLS'ens robusthets og sanntids egenskaper i et system. I slike systemer kombineres PC og PLS i en enhet hvor de kan kommunisere sammen. Slike systemer åpner for å bruke langt flere programmeringsspråk som gjerne har andre egenskaper enn de gitt av IEC 61131-3 standeren. Et eksempel på dette er maskinsyn. Det krever mange matematiske operasjoner som er vanskelig å implementere inn i språkene angitt av IEC 61131-3. Ved å bruke en PC i systemet åpner dette for bruk av språk som C++, Python, MATLAB og andre høynivå språk.

2.2.2 Beckhoff

Beckhoff Automation ble etablert i 1980 og leverte sin første PC baserte kontroller 6 år senere. Siden den tid har Beckhoff stått for en rekke oppfinnelser innen automatiseringsteknikk som Lightbus, busklemmene, EtherCAT og automatiseringsprogramvaren TwinCAT [17]. Selskapet har en raskt voksende tilstedeværelse og er med sitt verdensomspennende samarbeid tilstede i mer enn 75 land. Beckhoff leverer en rekke løsninger og er representert i industrier som maskin konstruksjon, prosess, vind, bygg automasjon, medisin og partikkel akseleratorer.

Beckhoff's CX serier er en modulbasert Embedded-PC. CX serien kombinerer PC og PLS teknologi i en pakke og kommer med en rekke konfigurasjoner og I/O moduler. Dette gjør serien skalerbar til mange forskjellige applikasjoner. CX-serien kommer med Windows operativsystem. CX2040 har en fire kjernet Intel Core i7 CPU, 4GB RAM.

Figur 2.12 viser internoppbygging av CX serien og hvordan PLS og PC kommuniserer. Sanntid (engelsk: real time, forkortet RT) operasjoner og ikke sanntid (engelsk: Non-Real Time forkortet N-RT) operasjoner kan kjøres samtidig og kommuniserer gjennom ADS protokollen. N-RT delen som kjører Windows kan bruke alle interne og eksterne Windows baserte biblioteker. RT delene er todelt med en PLS og en C++ del. C++ bibliotekene som er tilgjengelig i RT delen er veldig begrenset, og det er ikke mulig å bruke utenforstående biblioteker i denne delen.



Figur 2.12 Beckhoff internstruktur

Beckhoff har utviklet TwinCAT som står for The Windows Control and Automation Technology. TwinCAT er en programvare som brukes til å utvikle kontrollsystemer for deres produkter. TwinCAT kan også konvertere nesten alle PC baserte systemer til RT systemer. TwinCAT støtter IEC 61131-3, C og C++ samt gir mulighet for samspill med MATLAB og Simulink for enklere utvikling. TwinCAT er gratis å laste ned men de krever lisens for kommersielt bruk. TwinCAT 3 er nyeste versjon av programvaren og er brukt i dette prosjektet for all PLS utvikling. Beckhoff kjører en åpen tilnærming når det kommer til dokumentasjon. Alt av produkt og programvare dokumentasjon kan enkelt finnes på deres nettside [17].

2.3 Kommunikasjon

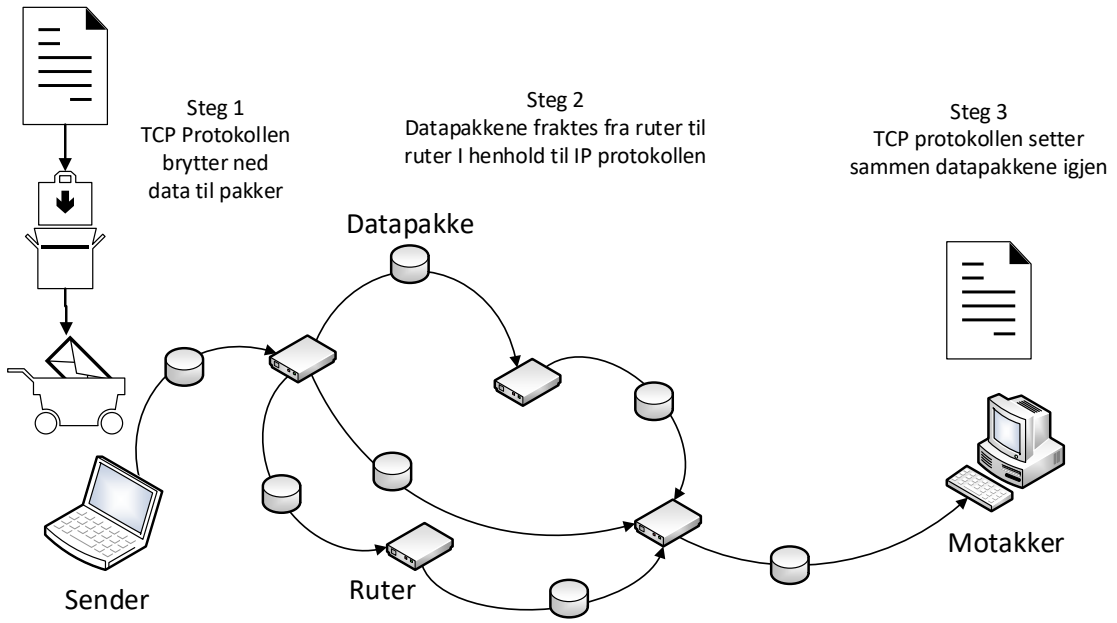
Dette kapittelet forklarer noen grunnleggende begreper innen data kommunikasjon.

2.3.1 Internett protokoller

"Internett Protocol" (IP) gjør det mulig å koble sammen flere forskjellige nett til et felles nett [18]. IP er en forbindelsesløst og upålitelig pakkeleveringstjeneste, men transportlags-protokollen "Transmission Control Protocol" (TCP) gjør det mulig å sende data pakker. TCP sikrer pålitelig transport av datapakker. I en TCP forbindelse er det en klient og en server. En klient "ringer" til serveren ved bruk av en IP-adresse. Serveren hører etter om noen "ringer" og kan enten ta imot samtalen eller avise den. Når samtalen er godtatt kan både server og klient sende beskjeder til hverandre. Samtalen er pågående til enten server eller klienten terminerer samtalen.

Figur 2.13 illustrerer det grunnleggende TCP/IP prinsippet hvor TCP protokollen bryter ned data til pakker og sender disse fra node til node i nettverket. Nettverket kan bestå av så lite som en sender og en mottaker med direkte kontakt eller mange sendere og mottakere med mange noder i form av rutere.

TCP/IP er en transportprotokoll som egner seg godt når en må være sikker på at beskjeden kommer fram og at det ikke er noe tap på veien. Denne sikkerheten gjør TCP/IP noe saktere enn andre protokoller som ikke har så streng oppfølging av kommunikasjonen. Det finnes alternativer til TCP da denne typen kommunikasjon ikke egner seg til alle typer applikasjoner.



Figur 2.13 TCP/IP

2.3.2 Andre protokoller

Automation Device Specification (ADS) er et transportlag innen TwinCAT systemet [19]. Protokollen ble utviklet for at data kunne utveksles mellom forskjellige programvaremoduler. ADS brukes blant annet for å kommunisere mellom N-RT og RT i Beckhoff produkter. Protokollen er ikke begrenset til internkommunikasjon innen TwinCAT miljøet. Om nødvendig kan det kommuniseres med andre PCer da ADS protokollen legger seg oppå TCP/IP eller UDP/IP protokollen. Dette betyr at i et nettverk system så er all data tilgjengelig fra ethvert ønskelig punkt.

ADS protokollen gir lese og skrive tilgang av data. For å få tilgang til variabler gjennom protokollen kan de kalles med adresse eller med variabelnavn. ADS har programmerings-grensesnitt (API) til følgende programmeringsspråk:

- C/C++
- .Net
- Delphi
- Java
- WebServices

2.4 Robot

2.4.1 ABB IRB 6600

ABB er en verdensledende leverandør av industri roboter og robotprogramvare, utstyr og komplette løsninger [20]. Med 300 000 roboter er de representer i de fleste næringer. ABB's roboter er brukt i alt fra plukke- og plasseringsløsninger innen matvare og legemiddel industrien til sveise, maling og sammenstillings løsninger innen automobil industrien. ABB leverer alt fra industri roboter ment for tunge og raske operasjoner til robotløsninger som skal samhandle med mennesker.

ABB IRB 6600 -175/2.55 vist i Figur 2.14 er en industri robot ment for punktsveising, materialhåndtering og maskin mating. Den har en stasjonær base, maks løftekapasitet på 175kg og maks rekkevidde på 2.55m. Roboten har en posisjons repeterbarhet på $0.1mm$ og bane repeterbarhet på $1.88mm$ [21]. ABB IRB 6600 leveres med IRC5 robotkontroller som gjør det mulig å kjøre roboten manuelt via en håndholdt kontroller, Flexpendant ,eller programmeres ved bruk av høynivå språket Rapid.

RobotStudio er ABBs verktøy for offline og online programmering av deres roboter. Offline programmering gjør det mulig å simulere robot, kontroller og dens oppgaver uten å ha direkte tilgang på roboten. Et visuelt brukergrensesnitt gjør RobotStudio enkel i bruk. For at roboten skal kunne utføre handlinger spesifisert i RobotStudio konverteres dette til Rapid kode.

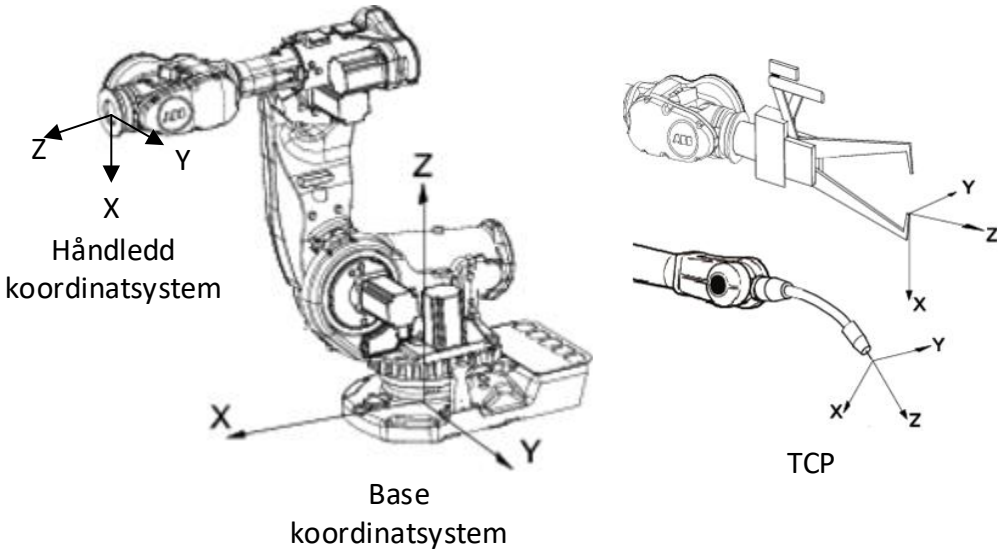
Figur 2.14 ABB IRB 6600

2.4.2 Kalibrering av verktøy

Når en robots bevegelse er programmert ved å spesifisere en bane eller posisjoner som roboten skal følge er dette normalt i forhold til verktøyets senter. Verktøyets senter forkortet VS i resten av avhandlingen (engelsk: tool centerpoint TCP). Normalt er VS definert som det aktive punket til verktøyet som senter av et gripeverktøy eller munnstykket til en sveise verktøy. Flere VS kan defineres men bare et kan være aktivt om gangen. Når et VS er aktivert vil robotarmen prøve å følge den gitte banen og treffe de ønskede punkt med VS.

For at robot armen skal kunne treffe ønskede posisjoner, brukes et kartesisk koordinatsystem. VS flyttes i forhold til base koordinatsystemet eller andre predefinerte koordinatsystemer. Kontrollsystemet kan da ved kinematikk og inverskinematikk regne ut vinkelhastighet og vinkelposisjonene til hvert enkelt led for å oppnå ønsket posisjon. Matematikken bak kinematikken og inverskinematikken er forklart i detaljer i "*Modelling and Controll of Robot Manipulators"* [22].

Base koordinatsystemet definerer posisjonen og orienteringen til roboten. Som standard er base koordinatsystemet likt verdenskoordinatsystemet. Håndleddets koordinatsystem definerer posisjonen til handleddet i forhold til basen. Verktøyet som skal utføre ønskede operasjoner er montert på håndleddet og får sitt eget koordinatsystem. Dette koordinatsystemet definerer VS er i forhold til håndleddet. Figur 2.15 viser de forskjellige koordinatsystemene.



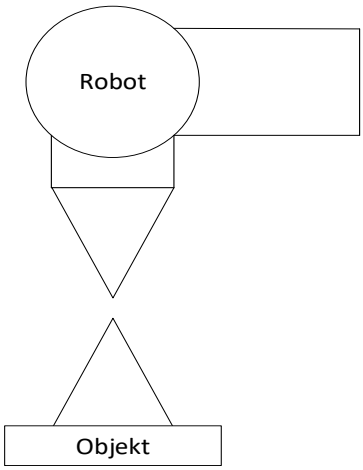
Figur 2.15 Koordinatsystemer [23]

Verktøy som monteres på robot må kalibreres og VS defineres ut i fra kjente referanser. Det finnes en rekke metoder for å kalibrere VS og det er en rekke fordeler og ulemper med disse.

3D-Model kalibrering

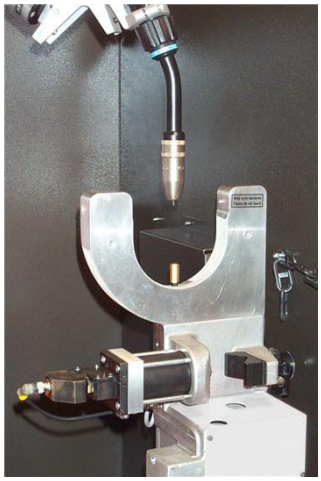
Det er mulig å finne VS ut i fra 3D-modellen. Dette er en enkel og billig metode. Det er stor usikkerhet med tanke på nøyaktighet ved denne metoden da den er avhengig av høy nøyaktighet ved produksjon av verktøyer. Det er ofte avvik fra 3D-modeller og den produserte komponenten. Høy produksjonsnøyaktighet fører også til høye kostnader.

Manuell Kalibrering
Ved manuell kalibrering beveges roboten til et referansepunkt. Dette gjøres flere ganger fra forskjellige vinkle og robotens VS kan kalkuleres. En vanlig måte er å bruke et spisst referansepunkt på verktøyet og objektet som vist i Figur 2.16. Denne metoden er kost effektiv og enkel men resultatet er operatøravhengig. Kalibreringsmetode egner seg til applikasjoner som ikke krever høy nøyaktighet.



Figur 2.16 Manuell kalibrering

ABB Bulls Eye kalibrering
Denne teknologien er basert på laser og kan nøyaktig finne X og Y-akse i verktøyets koordinatsystem. Kalibreringsmetoden virker ved å bevege verktøyet gjennom en laser se Figur 2.17. Denne metoden har begrensninger da den bare fungerer på noen typer verktøy og er en relativ kostbar metode.



Figur 2.17 Bulls Eye Kalibrering

For flere metoder se *"Method for calibration of off-line generated robot program"* [24] og "*Robot Tool Centre Point Calibration using Computer Vision"* [25]

2.4.3 Øye-hånd kalibrering

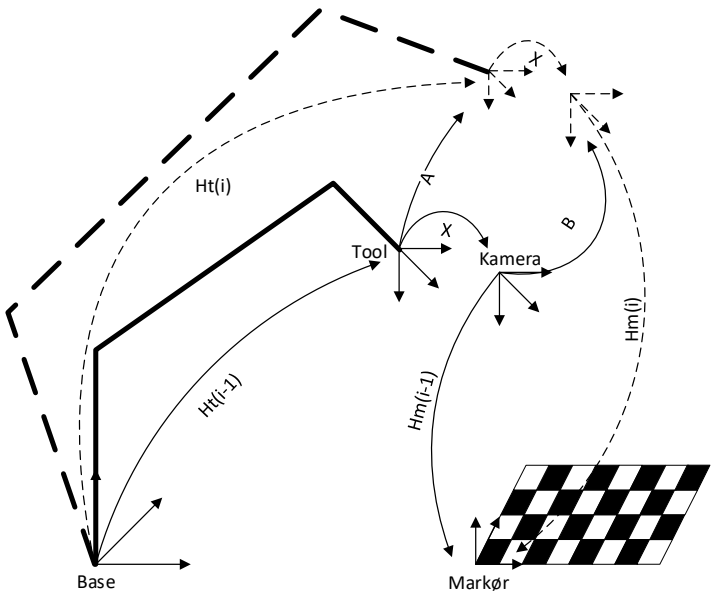
Et problem som ofte oppstår inne robotteknikk er å definere den relative posisjonen av sensor i forhold til håndleddet eller VS. For å finne den relative posisjonen brukes ligning 9 [26]. I denne avhandlingen er det snakk om kamera som sensor. A beskriver posisjon og orientering av håndleddets koordinatsystem i forhold til seg selv etter en vilkårlig bevegelse. B beskriver posisjon og orientering av sensorens koordinatsystem i forhold til seg selv etter den samme bevegelsen. X beskriver posisjon og orientering av sensor relativ til håndleddet.

$$AX = XB \tag{9}$$

For å finne A og B brukes ligning 10 der i beskriver nåværende bevegelse og $(i-1)$ beskriver foregående bevegelse. H beskriver den homogene transformasjonsmatrisen, dette er beskrevet i neste kapittel. H_t er transformasjonene mellom robotens base til robotens verktøy, denne hentes ut fra robotens kontroller. H_m beskriver transformasjonen mellom kamera og markøren, denne finnes ved å bruke et kalibreringsmønster som gir kjente posisjoner i bilde og verden. H_m og H_t er illustrert i Figur 2.18

$$\begin{aligned} A_i &= \left(H_{t(i-1)}\right)^{-1}\left(H_{t(i)}\right) \\ B_i &= \left(H_{m(i-1)}\right)\left(H_{m(i)}\right)^{-1} \end{aligned} \tag{10}$$

Kalibrering involverer flere vilkårlige bevegelser for å finne en så presis løsning for X som mulig. Kamera utsatt for støy som kan påvirke nøyaktigheten. Så en praktisk tilnærming er å utføre mange målinger og finne en X som minimerer observerte feil. Detter er beskrevet i detaljer i Robot Sensor Calibration: Solving AX = XB on the Euclidean Group [26].



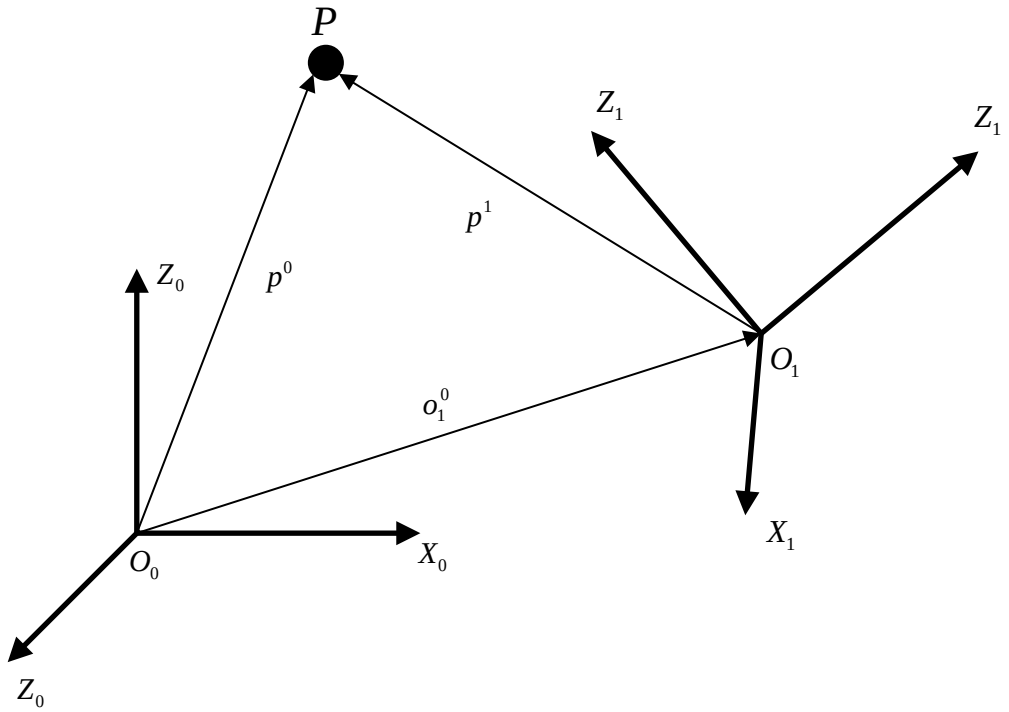
Figur 2.18 Øye hånd kalibrering

2.5 Kinematikk

Et fundamentalt krav innen robotikk og maskinsyn er å representere posisjon og orientering av objekter i rommet. Dette kapittelet forklarer noen grunnleggende prinsipper for å matematisk representere posisjon og orientering. Teorien og notasjonene for dette kapittelet er hentet fra [22], [2].

2.5.1 Homogen transformasjon

Et punkt i rommet kan bli representert av en koordinat vektor. Vektoren representerer forflytning av punktet med respekt til et referanse koordinat systemet. Koordinatsystemet brukt her er Kartesisk som er et sett med ortogonale akser som krysser i origo. Koordinatsystemet er merket med O_0 i dette tilfellet og aksene får notasjon X_0, Y_0 og Z_0



Figur 2.19 Representasjon av punkt P i forskjellige koordinat rammer

Det er antatt at punkt P, vist i Figur 2.19, er posisjon til et stivt objekt og representerer et punkt på et objektet. p^0 representerer romvektoren til punkt P med hensyn til koordinat ramme O_0. O_1^0 er en vektor som representerer koordinatsystem O_1 med hensyn til O_0. Vektor p^1 beskriver punkt P i forhold til O_1. Punkt P kan med dette bli representert i forhold til referanse ramme O_0 ut i fra ligning 11. Denne ligningen tar hensyn til rotasjon og translasjon. R_1^0 representerer rotasjonen av koordinatsystem O_1 i forhold til koordinatsystem O_0. Rotasjonsmatrisen blir forklart mer i detaljer senere i kapittelet.

$$p^0 = O_0^1 + R_1^0 p^1 \tag{11}$$

En mer handfast måte å beskrive rotasjon og translasjon av et koordinatsystem er ved homogen transformasjonsmatrise vist i ligning 12.

$$\tilde{p} = \begin{bmatrix} p \\ 1 \end{bmatrix} \tag{12}$$

Den homogene transformasjons matrisen A_0^1 i ligning 13 beskriver rotasjon og translasjon av koordinatsystem O_1 i forhold til koordinatsystem O_1 og kan bli skrevet som en 4x4 matrise.

$$A_1^0 = \begin{bmatrix} R_0^1 & O_0^1 \\ 1^T & 1 \end{bmatrix} \tag{13}$$

I homogen form kan punktet P bli forklart av den homogene vektoren $\tilde{p}^0$ som vist i ligning 14.

$$\tilde{p} = A_1^0 \tilde{p} \tag{14}$$

Kort oppsummert forklarer den homogene transformasjonsmatrisen koordinat transformasjonen mellom to koordinatsystemer i en kompakt form. Denne representasjonen kan bli utvidet til å beskrive et punkt P ut i fra en rekke med koordinattransformasjoner som vist i ligning 15.

$$\tilde{p}^0 = A_1^0 A_2^1 \ldots . A_n^{n-1} \tilde{p}^n \tag{15}$$

2.5.2 Rotasjonsmatrise og Euler vinkler

Representasjon av rotasjon i rommet er essensielt innen robotikk. Eulers rotasjons theorem sier at enhver rotasjon kan beskrives som en sekvens av rotasjoner om forskjellige koordinat akser. Euler's rotasjons theorem krever rotasjon om tre akser slik at ingen to etterfulgte rotasjoner er om samme akse. Det er 12 forskjellige måter å rotere aksene på. Roll – Pitch - Yaw er en av disse sekvensene som er rotasjon om Z-aksen, φ, etterfulgt av rotasjon om Y-aksen, ϑ, og tilslutt rotasjon om X-aksen, ψ. Den resulterende rotasjonen er beskrevet i ligning 16

$$R(\phi) = R_z(\varphi)\, R_y(\vartheta)\, R_x(\psi) \tag{16}$$

Ut ifra [22] kan rotasjonsmatrisen utrykkes ved ligning 17 når vinklene φ, ϑ og ψ er kjent.

$$R(\phi) = \begin{bmatrix} C\varphi C\vartheta & C\varphi S\vartheta S\psi - S\vartheta C\psi & C\varphi S\vartheta C\psi + S\varphi S\psi \\ S\varphi C\vartheta & S\varphi S\vartheta S\psi - C\vartheta C\psi & S\varphi S\vartheta C\psi - C\varphi S\psi \\ -S\vartheta & S\vartheta S\psi & C\vartheta C\psi \end{bmatrix} \tag{17}$$

Et av problemene med å uttrykke rotasjon på en slik minimalistisk måte er at singulariteter kan oppstå. Dette skjer når aksen av det midtre utrykket, $R_y(\vartheta)$, i ligning 16 blir parallell med rotasjonsaksen av det første, $R_z(\varphi)$, eller tredje utrykket, $R_x(\psi)$. Dette fører til tap av en frihetsgrad og matematisk betyr dette at ligning 17 ikke kan inverteres og da kan det bare etableres et lineært forhold mellom to av aksene.

$$R = \begin{bmatrix} R_{11} & R_{12} & R_{13} \\ R_{21} & R_{22} & R_{23} \\ R_{31} & R_{32} & R_{33} \end{bmatrix} \quad (18)$$

Roll – Pitch – Yaw vinkler opplever problemet med singularitet når rotasjon om Y-aksen er 90 grader. For å gå fra rotasjonsmatrisen tilbake til Euler vinkler brukes ligning 19 mellom -90 grader og 90 grader. Ligning 18 viser elementene i rotasjonsmatrisen.

$$\begin{aligned} \varphi &= Atan2\ (r_{21}, r_{11}) \\ \vartheta &= Atan2\left(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}\right) \\ \psi &= Atan2(r32, r33) \end{aligned} \quad (19)$$

Ligning 20 brukes i intervallet mellom 90 grader og 270 grader for å finne Euler vinklene.

$$\begin{aligned} \varphi &= Atan2\ (-r_{21}, -r_{11}) \\ \vartheta &= Atan2\left(-r_{31}, -\sqrt{r_{32}^2 + r_{33}^2}\right) \\ \psi &= Atan2(-r32, -r33) \end{aligned} \quad (20)$$

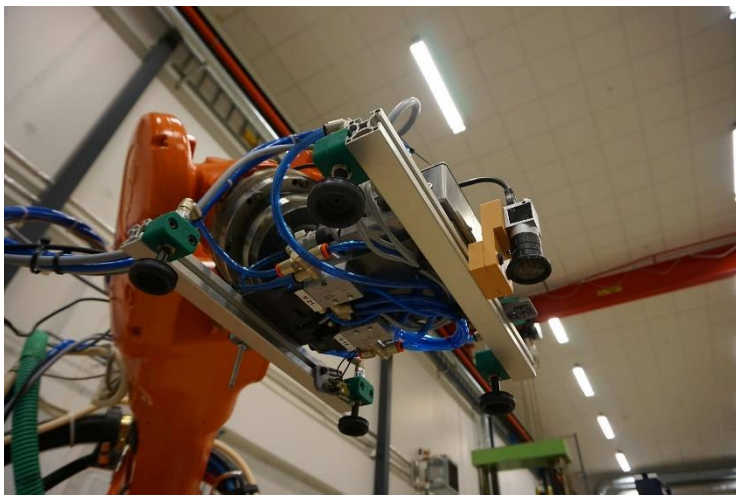
Ligning 19 og 20 kan ikke brukes dersom vinkel ϑ er nøyaktig 90 elle -90 grader. I dette tilfellet er det bare mulig å bestemme summen eller forskjellen mellom φ og ψ.

3 Metode

3.1 System

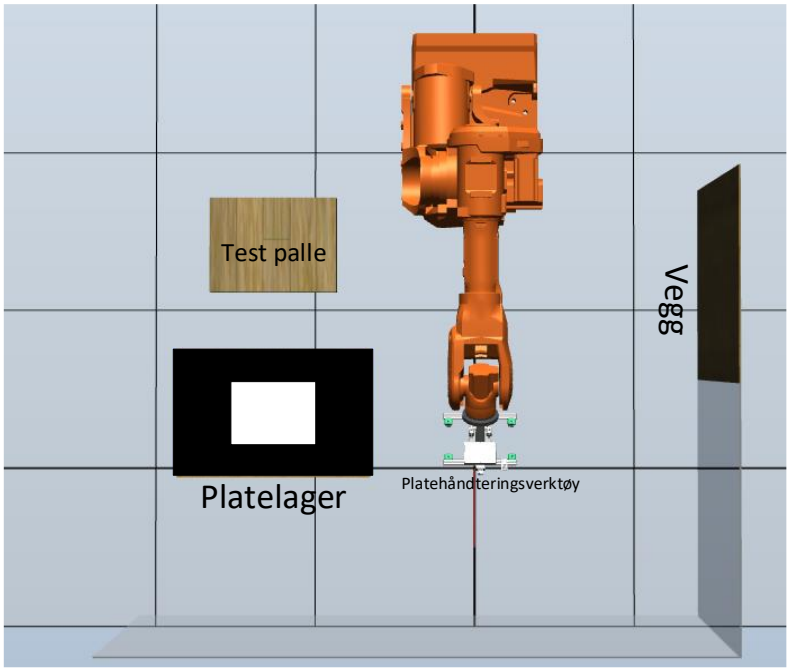
Dette kapittelet er ment for å gi et raskt overblikk over systemet. De forskjellige elementene i systemet vil bli forklart i mer detalj senere i avhandlingen. Det er laget en video [27] som beskriver systemet også.

Denne avhandlingen bygger videre på noe av arbeidet som ble utført i bacheloroppgaven *"Utvikling av verktøy for håndtering og montering av fasadeplater med robot"* [28] som var gitt ved UiA våren 2017. Verktøyet for plåtehandtering vis i Figur 3.1 blir gjenbrukt med oppgradert kamera. Verktøyet bruker vakuum for å plukke opp platene.



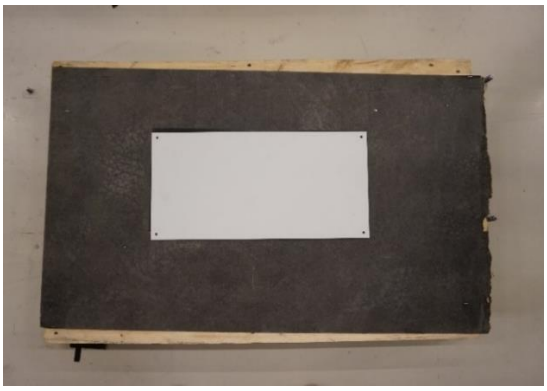
Figur 3.1 Platehåndterings verktøy

Robotcellen er bygget opp som vist i Figur 3.2. En ABB IRB6000 1.75/2.55 industri robot med tilhørende IRC5 kontroller brukes som manipulator arm. En palle dekket med svart papp avgrenser platelageret, se Figur 3.3. Platene kan ligge hvor som helst innenfor denne avgrensningen og roboten skal kunne lokalisere og plukke den opp. Dimensjon på platelageret er 1200mmx800mm. En mindre palle plassert ved siden av platelageret ble brukt under testing ved å flytte platene fra ene til andre pallen.



Figur 3.2 Robotcelle

Inne i robotcellen er det en vegg som platene skal monteres på ved å bruke maskinsyn og roboten. Det ble montert flatjern for å simulere stendere, se Figur 3.4. Magnettape ble montert på platene slik at de henger på stenderne. Platene brukt er av typen Steni Color og dimensjonene er 595x295x6mm.



Figur 3.3 Platelager

Figur 3.4 Vegg

Kontrollsystemet er en Embedded-PC av typen Beckhoff CX2040. Denne håndterer I/O signalene fra verktøyet samt utfører bildebehandlingen. CX2040 kommuniserer og sender ønskede koordinater til ABB IRC5 kontrolleren. IRC5 kontrolleren fungerer som en slave, den melder hvor roboten er i rommet og utfører det CX2040 kontrolleren forteller den. All kode som er brukt i dette prosjektet finnes i vedlegg D.

3.2 Maskinsyn

Dette kapittelet beskriver hvilke kamera og linse som ble valgt for oppgaven, hvordan kamera ble kalibrert og hvordan maskinsyn-programmet ble bygget opp.

3.2.1 Kamera og linse

Et av de rapporterte utfordringene ved det foregående prosjektet var unøyaktig maskinsyn [28]. Kameraet som ble brukt tidligere var et Logitech C260 med en oppløsning på 1.2Mp [29]. Dette er et web-kamera ment for konsumer bruk. Kriteriene for et nytt kamera var høyere oppløsning, mulighet for å hente ut ukomprimert bilder og USB eller Ethernet tilkobling.

UiA hadde et kamera av typen Basler acA2500-14um tilgjengelig. Dette kameraet har 5Mp oppløsning, USB3 tilkobling og utskiftbar linse, se Tabell 3.1 for flere spesifikasjoner. Kameraet tar enkeltfarge (engelsk: monochrome) bilder og gir mulighet for å hente ut ukomprimerte bilder. Dette kameraet er bygget for industrielt bruk og maskinsyn. Kamera kommer med eget programmerings-grensesnitt som gir bruker kontroll over flere parametere en et web-kamera.

Basler acA2500-14um	
Sensor størrelse	$1/2.5'$
Oppløsning (H x B)	$2592\ x\ 1944$
Oppløsning	$5\ Mp$
Sensor størrelse	$5.7mm\ x\ 4.3mm$
Piksel størrelse	$2.2\mu m\ x\ 2.2\mu m$
Lukker	Rullende
Datagrensesnitt	USB 3

For flere spesifikasjoner [30]

Tabell 3.1 Basler aAC2500 kamera

Arbeidsavstand for kamera på 0.5m – 1m ble valgt og linse måtte velges ut i fra dette. Valget falt på en TechSpec 4mm linse. Med denne linsen kan kamera se hele platelageret fra 1m og hele platen fra 0.5m. Linsen har 17.5% forvrenging over hele bilde [31]. Dette ble korrigert for under kalibrering.

Kamera ble montert midt på verktøyet, se Figur 3.1. En brakket som kunne monteres på aluminiumsprofilene ble 3D-printet. Dette gjør det mulig å flytte kamera rundt på verktøyet om andre konfigurasjonen ønskes eller nytt utstyr skulle bli montert. Plasseringen av kamera ble valg for enkel implementering og unødvendige bevegelser under plukke/ plasserings-prosessen.

3.2.2 Kamera kalibrering

Kamera som brukes til maskinsyn må kalibreres for å finne indre og ytre kamera parametere. På grunn av stor geometrisk forvrenging på linsen valgt var det også nødvendig og finne forvrengnings koeffisientene.

Kalibrering bør gjentas dersom følgende gjøres:

- Endring av fokus, fører til små forandringer i fokal lengde.
- Ending av blenderåpning
- Om linsen skrues av og på

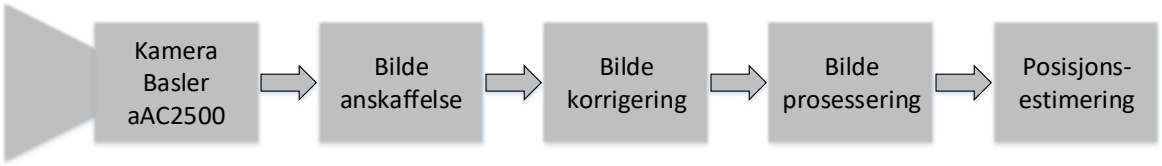
Et sjakkbrett mønster bestående av 9x6 ruter, gitt av OpenCV.org [3] brukes ved kalibrering. A3 format gir kvadratiske ruter på 37mm som blir brukt til å estimere kameras indre og ytre parametere samt geometrisk forvrenging. Rutenettet ble montert på en av fasadeplatene for å gi en plan overflate. Under kalibrering var mønsteret stasjonært mens kamera ble flyttet rundt ved bruk av roboten.

Det ble tatt bilde av sjakkbrett mønsteret fra flere forskjellige vinkler og over hele kameraets synsfelt. For å analysere alle bildene og få ut alle nødvendige parametere ble MATLABs kamera kalibrerings vektøyskasse brukt. Det er et brukervennlig verktøy og lar brukeren bestemme hvilke parametere som skal estimeres. Andre verktøy som ble vurdert for å estimere de nødvendige parameterne var OpenCV. Det finnes ferdige eksempler på OpenCV.org men disse er ikke like brukervennlig da det er C++ eller Python kode og ikke et enkelt brukergrensesnitt som i MATLAB.

3.2.3 Bilde behandling

I denne avhandlingen ble det spesifisert at all bildebehandling skulle gjøres ved bruk av OpenCV og kodes i C++. For å utvikle maskinsyn programmet i denne avhandlingen ble det valgt å bruke Microsofts utviklingsverktøy Visual Studio Community 2015. Dette er en gratis utgave av utviklingsverktøyet til Microsoft. Dette er et velprøvd verktøy for utvikling innen C++ med et stort nettsamfunn med mye dokumentasjon. Visual Studio må konfigureres for og bruke OpenCV og andre biblioteker som ikke er utviklet av Microsoft [32].

Figur 3.5 viser maskinsynets oppbygging i grove trekk for bildeprosessering og posisjonsestimering. Alle delene av prosessene vil bli forklart. OpenCV funksjoner brukt i dette kapittelet blir beskrevet med `cv::funksjonsnavn,` og siden OpenCV er basert på det engelske språk vil `funksjonsnavnet` stå på engelsk



Figur 3.5 Hovedoppgaven til maskinsyn

Bilde anskaffelse

OpenCV biblioteket har innebygde metoder å hente ut bilde fra webkamera. Siden et Basler industrikamera brukes her, må bilde hentes ut via Pylon API [33] for så konverteres til OpenCV matrise format. Pylon er kamera programvare brukt av Basler. Basler kamera kan levere Mono8 eller Mono12 bildeformat. OpenCV har direkte støtte for 8,16 og 32 bit format. For å ikke gjøre noen spesielle operasjoner ble det valg å bruke 8 bit format for enkel konvertering fra Pylon til OpenCV.

Bildekorrigering

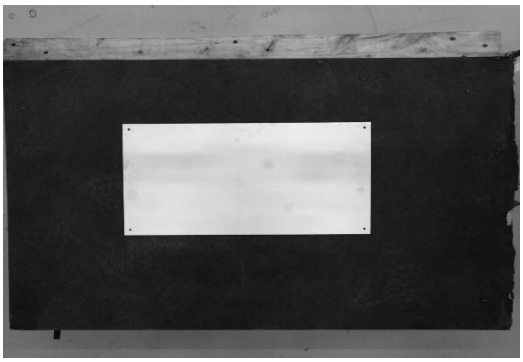
For å korriger for kameraets geometriske forvrenging brukes ferdige OpenCV funksjoner. Funksjonen `cv::undistort` baserer seg på teorien i kapittel 2.1.3. Funksjonen trenger følgende inndata: et bilde i OpenCV `cv::Mat` format, interne kamera parametere, radiell forvrenging koeffisienten, tangentielle forvrenging koeffisientene.

Bildeprosessering

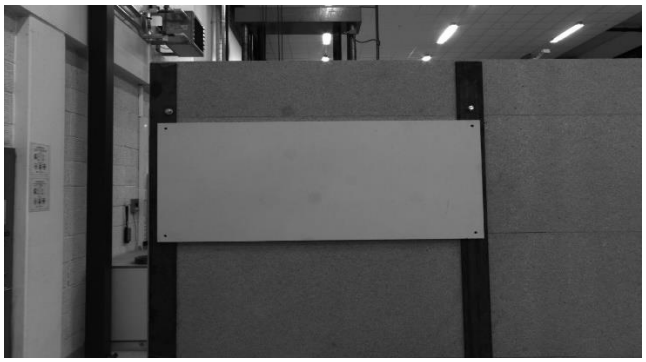
Det er mange måter å hente ut ønskede konturer fra et bilde. OpenCV har en hel del algoritmer for å gjennomføre denne prosessen. Det var ønskelig å gjennomføre denne prosessen på minst mulig komplisert måte. Grunnen til dette er at testoppsettet er innendørs under kontrollerte forhold med stabilt lys. Det var ønskelig å vise at dette var mulig å gjøre bildeprosesseringen på en enkel måte slik at det er flere valgmuligheter når forholdene blir vanskeligere.

To forskjellige bildeoperasjoner skal gjøres. Når platen skal plukkes opp må dens posisjon estimeres. Når platen skal monteres på veggen skal den plasseres ut i fra en allerede montert plate. For å kunne gjøre dette må posisjonen til den allerede monterte platen estimeres. Dette fører til at konturene av platen må finnes under to forskjellige forhold. For å skille platen fra bakgrunnen må noen operasjoner gjøres.

Når platen ligger på platelageret, vist i Figur 3.6, kommer alt lyset ovenfra. For å skille bakgrunnen fra platen blir det brukt en enkelt grenseverdi funksjon, `cv::thresholding`, funksjonen baserer seg på teorien forklart i kapittel 2.1.4 .



Figur 3.6 Plate ved platelager

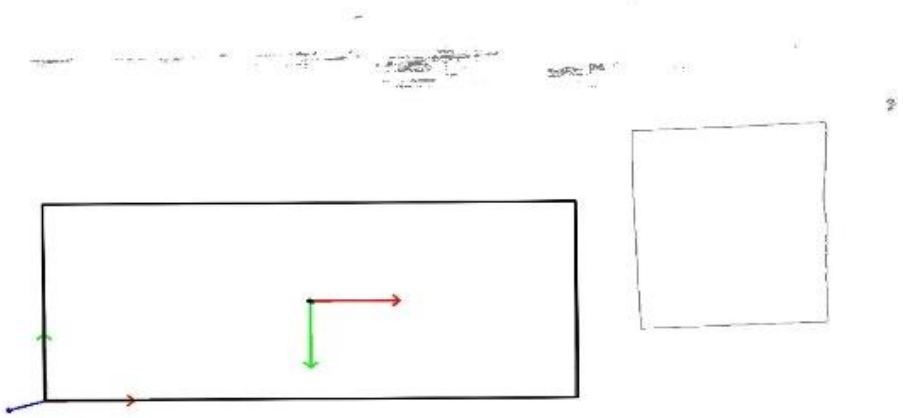


Figur 3.7 Plate på vegg

Figur 3.7 viser platen på veggen. Det kan observeres litt vanskeligere forhold her. Noe av lystet kommer rett ovenfra og noe av lystet kommer skrått. På venstre siden av bilde kan det observeres at bakgrunnen er mye samme farge som platen. Dette kan løses på flere måter men i dette tilfellet ble det valgt og bruke et slør filter og Otsu grenseverdi metoden for å beregne passelig grenseverdi. OpenCV funksjonene brukt ved veggen var `cv::GaussianBlur` og `cv::threshold(THRESH_BINARY | THRESH_OTSU)`.

Neste operasjon er å finne platens kontur. Dette ble gjort i to steg. Først ble alle konturene i bilde funnet ved å bruke funksjonen `cv::findcontoures`. Denne funksjonen baserer seg på teorien i kapittel 2.1.5 og finner alle konturene i bildet.

Andre steget er å skille mellom andre konturer og platen. Siden platen er formet som et rektangel brukes funksjonen `cv::RotatedRect`. Denne funksjonen finner alle rektangler i bilde. Funksjonen kan finne rektangler som ikke er platen og for å løse dette ble det antatt at det ønskete rektangelet er det med størst areal. Dette er illustrert i Figur 3.8 der platen ligger vedsiden av et mindre A4 ark og funksjonen finner og tegner svart rundt det rektangelet med størst areal. `cv::RotatedRect` funksjonen gav mye nyttig informasjon som hjørnepunktene, senterpunktet, høyde, bredde og areal for å nevne noen.

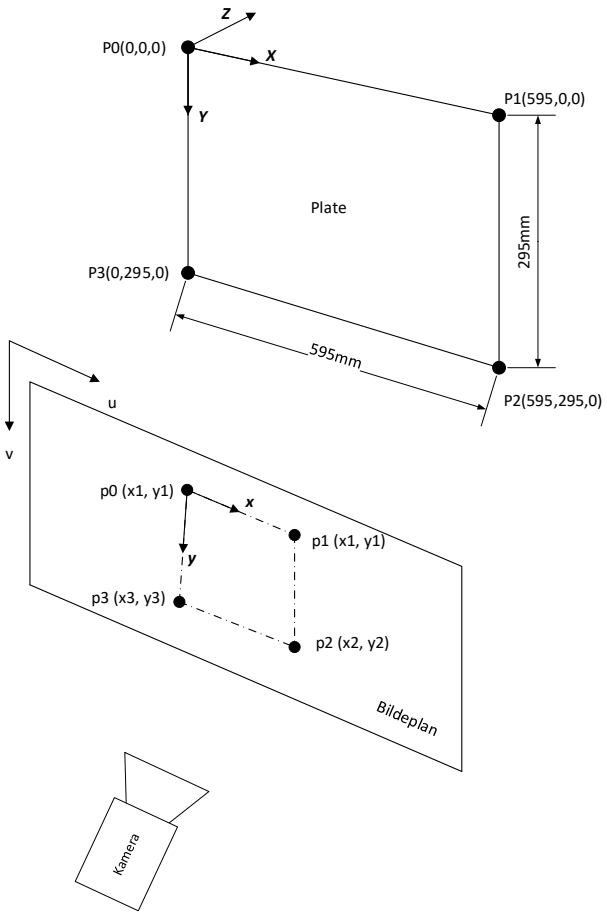


Figur 3.8 Platekonturer

Posisjonsestimering
Når det brukes et enkelt kamera forsvinner informasjon om distanse til objektet når det projiseres over i bildeplanet som forklart tidligere. For å motvirke dette ble to forskjellige metoder for posisjonsestimering utprøvd med varierende resultat.

Metode 1:
En ekstern avstandsmåler, en Lidar sensor måler avstand fra verktøyet til platen. Avstanden kan brukes for å finne Z koordinater til platen og X og Y kan da estimeres. LIDAR sensoren var montert på vakuum verktøyet [28]. Hvordan avstandsmålingen ble implementert og vurdert blir forklart senere.

Metode 2
Ved å bruke kjente dimensjoner i bildet kan posisjonen estimeres som et perspektiv-til-punkt (PnP) problem. I denne avhandlingen er fasadeplatens dimensjoner kjent og relativt konstant med en oppgitt nøyaktighet fra Steni på $\pm 2mm$ i lengde og bredde [34]. OpenCV har innebygget funksjoner for å løse PnP problemer. Platen har fire hjørne som er funnet i bildeprosesseringen. Platen ligger på et plant underlag så det kan antas at det er ikke er høydeforskjell mellom hjørnene. Ved å bruke ene hjørnet som nullpunkt kan koordinatene til de andre hjørnene beskrives som vist i Figur 3.9. Forholde mellom de fire hjørnene i bildeplanet og virkeligheten er da kjent og funksjonen `cv::solvPnP` brukes for å finne platens translasjon og rotasjon i forhold til kamera.



Figur 3.9 Posisjonsestimering

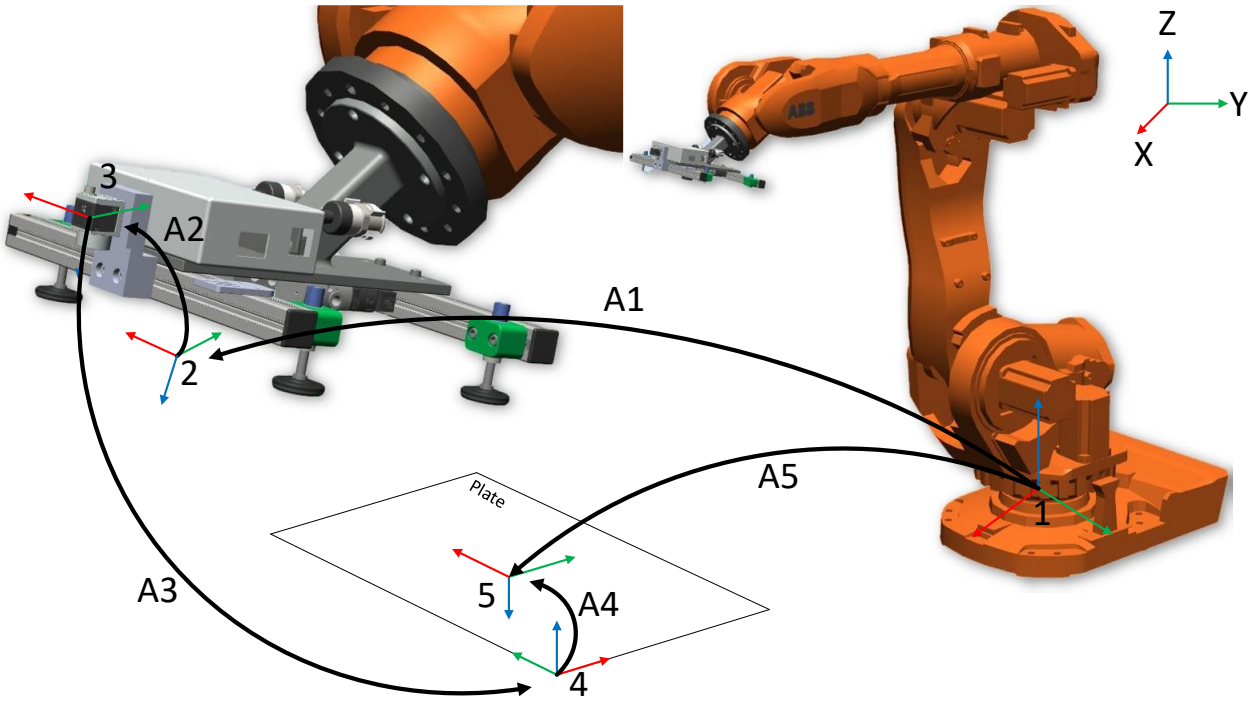
3.2.4 Kinematikk

For at roboten skal kunne plukke opp platen må platens translasjon og rotasjon beskrives i forhold til robotens base. Figur 3.10 viser hvordan dette er oppnådd. De forskjellige koordinatsystemene er merket med numre fra 1 til 5. 1 er robotbasens koordinatsystem, 2 er verktøyets senter, 3 er kameraets koordinatsystem, 4 er koordinatsystem i platens hjørne og 5 er platesenterets koordinatsystem.

Ligning 21 viser hvordan platens translasjon og rotasjon blir kalkulert.

- A_5 er transformasjonsmatrisen som beskriver platens translasjon og rotasjon i forhold til robotbasen
- A_1 beskriver translasjon og rotasjon av VS i forhold til basen og hentes fra IRC5 kontrolleren.
- A_2 beskriver translasjon og rotasjon av kamera i forhold til VS, dette er en fast verdi som finnes ved øye-hånd kalibrering.
- A_3 er translasjon og rotasjon av plates hjørnepunkt i forhold til kamera og blir estimert av maskinsynet og `cv::solvPnP` funksjonen.
- A_4 er translasjon og rotasjon av platens senter i forhold til hjørnet 4. Dette er også en fast verdi som er halve platens brede og halve platens lende. Platens koordinatsystem roteres slik at det peker i samme retning som kameraets koordinatsystem.

$$A_5 = A_1 A_2 A_3 A_4 \tag{21}$$

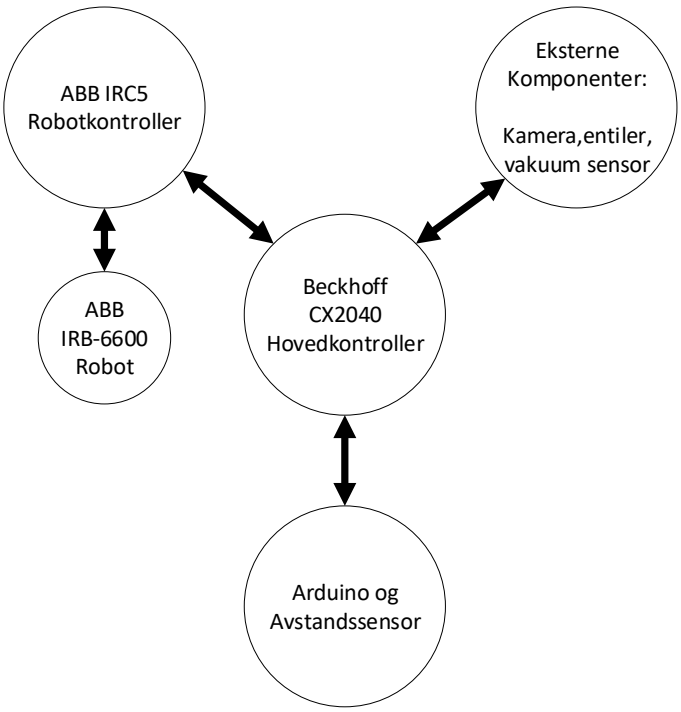


Figur 3.10 Systemets koordinatsystemer

Operasjonen med å regne ut platens posisjon og rotasjon i forhold til robot basen blir gjort i C++ og N-RT delen av kontrolleren som forklares under. Dette blir gjort fordi det blir benyttet et eksternt bibliotek som utfører matrise operasjoner. Denne prosessen er ikke RT sensitiv siden roboten står i ro under posisjonsestimeringen, derfor blir det valgt å utføre dette i N-RT delen av kontrolleren. Biblioteket Eigen brukes, dette er et C++ bibliotek for lineær algebra, matriser, vektorer og numeriske operasjoner. Eigen er åpen kilde kode og er lisensiert under MPL2 lisens.

3.3 Kontrollsystem

Dette kapittelet forklarer hvordan kontrollsystemet til roboten er satt opp og fungerer. Figur 3.11 gir oversikt over kontrollsystemet og hvordan hovedkontrollen kommuniserer med resten av systemet. Kontrollerens struktur forklares samt hvordan menneske-maskin grensesnittet (engelsk: human maskin interface, HMI) er bygget opp.



Figur 3.11 System oppbygging

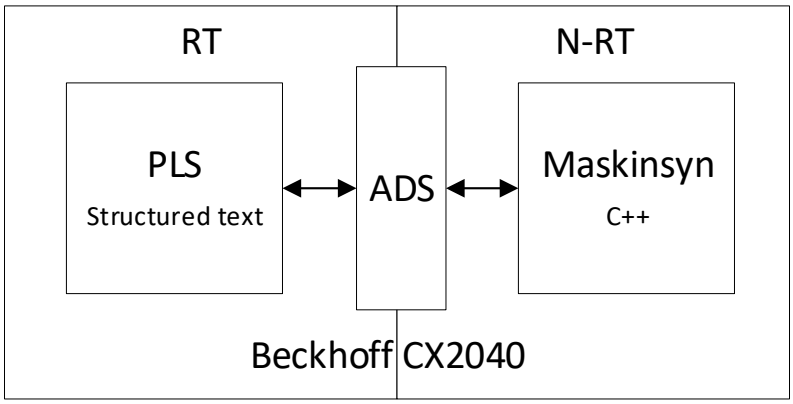
3.3.1 Implementering av CX2040 Embedded PC

I dette prosjektet ble det spesifisert at en Beckhoff CX2040 skulle brukes som hovedkontroller. Den skal plasseres vekk fra roboten kontra tidligere prosjekt da kontrolleren ble plassert i verktøyet. Alle I/O signaler og hovedfunksjonene skal plasseres i PLS.

Hovedkontrolleren i systemet skal gjør følgende:

- Styre alle I/O signaler.
- Kommunisere med IRC5 kontrolleren via TCP/IP.
- Kjøre en enkel HMI.
- Kommunisere med Basler aAC2500 kamera.
- Bildebehandling ved bruk av OpenCV.
- Kommunisere med Arduino for avstandsmåling.
- Kommunikasjon mellom PLS(RT) og Windows (N-RT).

I kapittel 2.2.2 ble det forklart hvordan CX2040 har en RT og NRT del. I dette prosjektet kjøres kun PLS og dens funksjoner i RT mens maskinsynet kjøres i NRT som vist i Figur 3.12.



Figur 3.12 CX2040 struktur

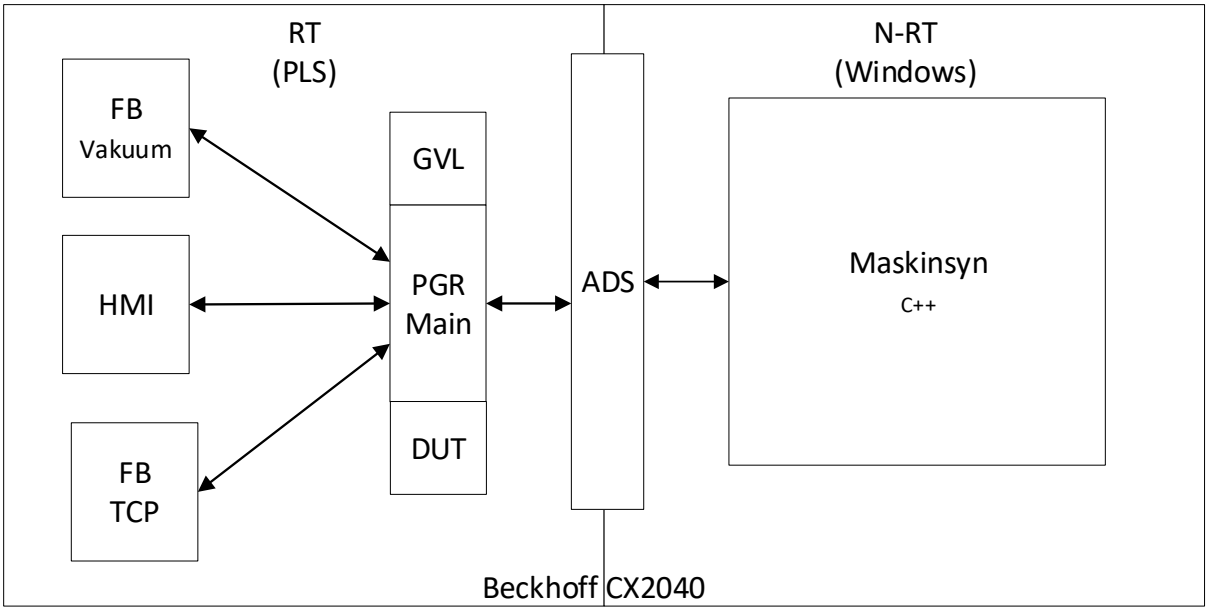
I/O Håndtering

Platehåndteringsverktøyet bruker vakuum for å plukke opp platene. Vakuum kretsen består av en 100 liters tank med vakuum pumpe, 2 x 3/2 veis magnetventil og 2 x trykksensorer og 4 x sugekopper. Vakuum kretsen er todelt og kan løfte platen selv når den ene kretsen er ute av drift, for illustrasjon av kretsen se [28]. En trykkbryter er montert på en av sugekoppen for å registrere om verktøyet kolliderer med platene.

CX2040 og modulene brukt i dette prosjektet er beregnet for 24V. Ventilene og trykk sensorene er også beregnet for 24V og kan enkelt kobles til kontrolleren. Se vedlegg B for I/O liste og komponent forklaring, se vedlegg C for koblingsskjema av systemet.

3.3.2 Oppbygging av PLS logikk

Utvikling av PLS programmet ble gjort i TwinCAT 3. Figur 3.13 viser blokkene som utgjør PLS programmet. Alle blokkene ble programmert ved bruk av Strukturert tekst(ST), dette ble brukt fordi dette er et høynivåspråk der flere operasjoner/instrukser kan utføres med enkle kommandolinjer. ST kan sammenlignes med andre høynivåspråk som Pascal eller C, og dette ble sett på som en fordel da C++ og Rapid som også brukes i dette prosjektet er basert på C.



Figur 3.13 Utvidet CX2040 struktur

Et PLS program består av en eller flere programorganiseringsenheter (engelsk: Program orginization uint, POU). Dette er måten en PLS holder orden. En POU blokk kan være en av tre typer:

- Program - returnerer en eller flere verdier ved utførelse. Etter et program er utført lagres alle verdiene fram til programmet blir utført på nytt.
- Funksjon - returnerer bare et data element ved utførelse. Data elementet kan være en rekke (engelsk: array).
- Funksjonsblokk - som returnerer en eller flere verdier ved utførelse.

Enhver POU kan kalle opp andre POU. Som regel vil et program inneholde kall/bruk av flere funksjoner og funksjonsblokker. Det er også mulig å kalle opp et program fra et annet program.

I denne avhandlingen er PLS programmet bygget opp med POU blokkene vist i Figur 3.13. Blokkene har forskjellig oppgaver og er forklart under:

- Main blokken er en program POU. Main blokken har høyest prioritet i PLSen og blir skannet hver syklus. Alle blokker blir tilkalt av Main programmet når det skal utføres. Maskinsyn programmet leser og skriver til variabler i Main blokken. Den inneholder også all systemlogikken, altså hva som skal gjøres og når.

- TCP blokken tilkalles av Main blokken for å utføres. Denne blokken håndterer all kommunikasjon med IRC5 kontrolleren. Kommunikasjonen forklares i kapittel 3.4.
- Vakuum blokken står for håndtering av I/O signalene fra verktøyet.
- GVL blokken er global variabel liste som inneholder variabler som brukes av flere blokker. Funksjonsblokkene og programblokkene inneholder også sine individuelle inngangsvariabler, utgangsvariabler og internvariabler.
- DTU blokken (engelsk: Data unit type) inneholder forskjellige datatyper som brukes til å lage forskjellig tilstander (engelsk: states).
- HMI blokken inneholder det grafiske grensesnittet operatøren ser. Denne blokken er grafisk programmert i TwinCAT 3's visualiseringsverktøy.

Main blokken
For å designe logikken til systemet måtte ønsket funksjonalitet beskrives, etter flere iterasjoner ble følgene funksjonalitet overveid som mest viktig.

- Åpne kommunikasjon med IRC5, sende og motta data.
- Bevege robot til hjem posisjon.
- Bevege robot til platelager.
- Bevege robot til test palle.
- Bevege robot til vegg.
- Ta bilde og estimere ny posisjon (platelager og vegg).
- Plukke opp eller plassere plate (platelager, test palle og vegg).
- Auto.
- Stopp.

For å gjøre systemet egnet for testing og feilsøking ble det bestemt at hver enkelt funksjonalitet kunne brukes manuelt samt kombineres for å skape automatiske funksjoner.

I starten var det meningen å ta med en snuoperasjon av platen, dette fordi platene blir levert slik at to plater ligger lakk mot lakk, da må ene platen snus før den kan plassers på veggen. På grunn av tidsbegrensinger ble det valgt og se bort fra denne operasjonen da den ikke er vital for å få testet maskinsynet.

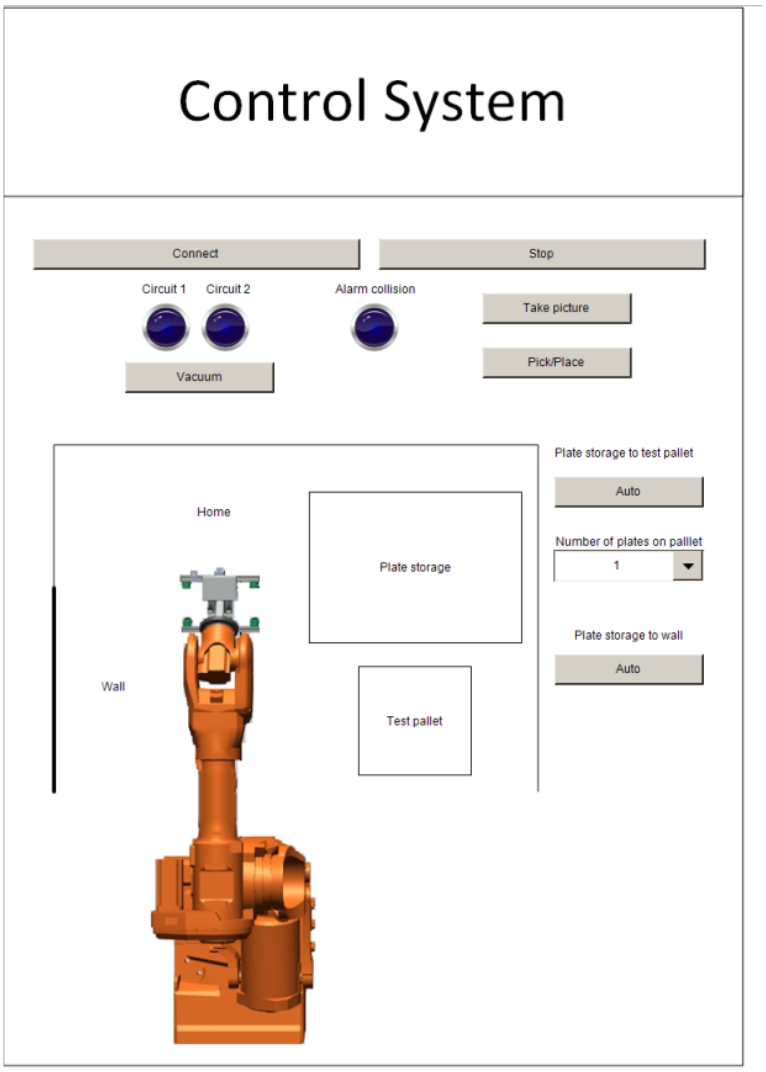
3.3.3 HMI

HMI er todelt i dette systemet, der den ene er laget i PLS programmet og den andre i maskinsyn programmet. HMIen laget i PLS programmet har et enkelt designe der bruker kan operere systemet men får minimalt med informasjon. HMIen i maskinsyn programmet gir informasjon om maskinsynets utførelse. Operatør kan ikke utføre operasjoner fra denne HMIen. Begge HMIene er designet ved brukt av engelsk, så engelske ord i dette kapittelet beskrive knapper, alarmer og tilstander.

HMI tilknyttet PLS

TwinCAT 3 gjøt det mulig å designe HMI med et dra og slipp grensesnittet eller HTML5. HMI kan også designes og kjøres fra N-RT delen av CX2040 kontrolleren. Dra og slipp grensesnittet ble valgt på grunn rask implementering og enkel bruk.

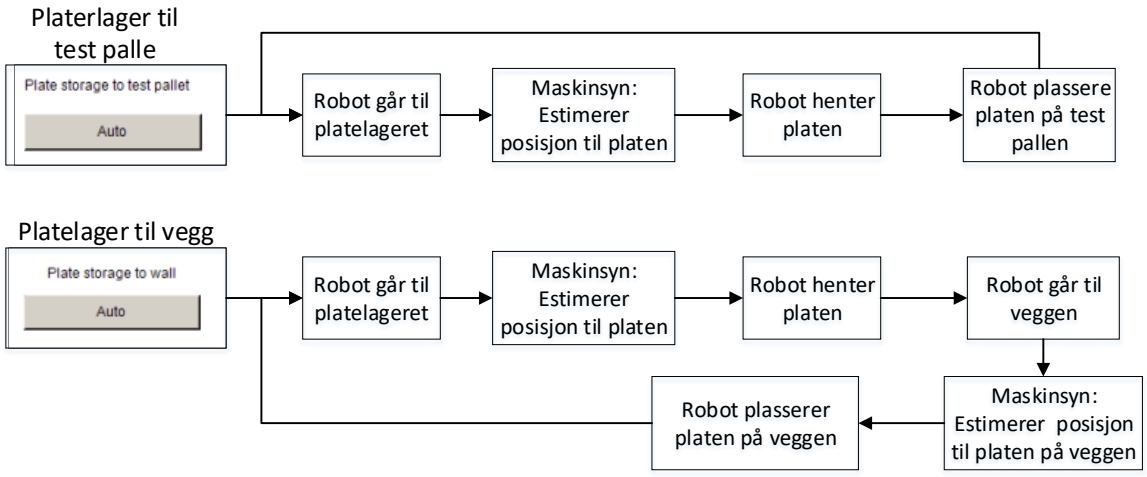
Figur 3.14 viser HMIen utviklet for denne avhandlingen. Den består av et vindu. HMIen er designet med en hvit bakgrunn for å gi god kontrast til farger som beskriver pågående prosesser og alarmer. Knappene er grå og lampene er blå når de ikke er aktiv. Knapper og lamper blir grønn når de er aktiv. Alarmer blir rød når de er aktiv. Robotcellen er vist i HMIen. Roboten er et stillbilde men sonene lyser grønn når de er aktiv.



Figur 3.14 HMI i PLS

Det er to "Auto" knapper. Den øverste brukes når platene skal flyttes fra platelageret til test pallen. Når denne skal benyttes må det spesifiseres hvor mange plater som ligger ved platelageret. Dette gjøres i en nedtrekks meny under knappen.

Den nederste "Auto" knappen brukes til å hente plater ved platelageret og plassere de på veggen. Systemet kan foreløpig bare plassere en plate på veggen derfor tar denne operasjonen kun en plate på platelageret. Flyten til "Auto" knappene er beskrevet i Figur 3.15.



Figur 3.15 Auto funksjon

Knapper

- Connect - Starter kommunikasjonen med IRC5. Knappen vil lyse grønn så lenge kommunikasjonen er opprettholdt.
- Vacuum – Bruker kan manuelt slå av og på vakuum. Bruker trenger ikke tenke på dette ved Auto operasjoner.
- Stop – Stenger kommunikasjonen med IRC5 kontrolleren.
- Take picture – Aktiverer maskinsynet.
- Pick/Place – Bruker kan manuelt plukke opp eller plassere plate. Det fungerer ikke å manuelt plukke/plassere med mindre "Take picture" er gjennomført.

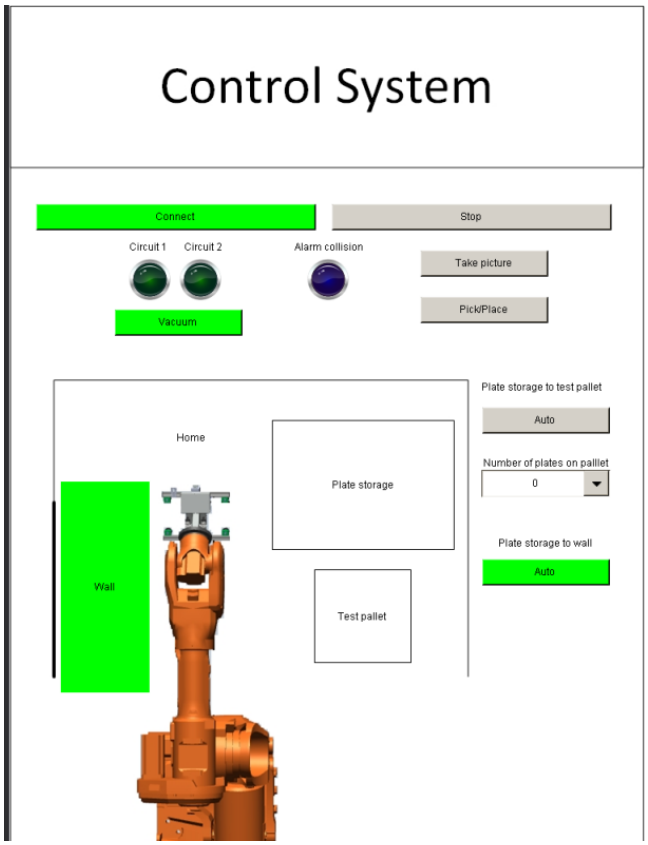
Lys

- Alarm collision – lyser rødt dersom trykk bryteren på verktøyet aktiveres, da er sugekoppene trykket for langt ned. Bruker trenger ikke gjøre noe da det er en sikkerhetskrets som går til IRC5 kontrolleren som stopper roboten.
- Circuit 1 and Circuit 2 – Lyser grønn når vakuum i kretsen er over 70%, som er tilstrekkelig for å løfte platen.

Figur 3.16 viser HMI mens den er i brukt. Følgende kan observeres:

- Connect – er aktiv.
- Auto (Plate storage to wall) – er aktiv
- Vacuum – er aktiv.
- Circuit 1 and Circuit 2 – er aktiv. Dette betyr at det er over 70% vakuum i begge kretsene. Vakuum verktøyet har da plukket opp en plate.
- Wall – er aktiv. Dette betyr at roboten er på vei til veggen. Når roboten har når posisjonen som er definert som veggen blir den hvit igjen.

I HMIen vist i Figur 3.14 er det er også mulig å manuelt flytte på roboten uten og bruke "Auto" knappene. Ved å trykke inni robotcellen på "Home", "Wall", "Test palet" og "Plate storage" går roboten til disse posisjonene.



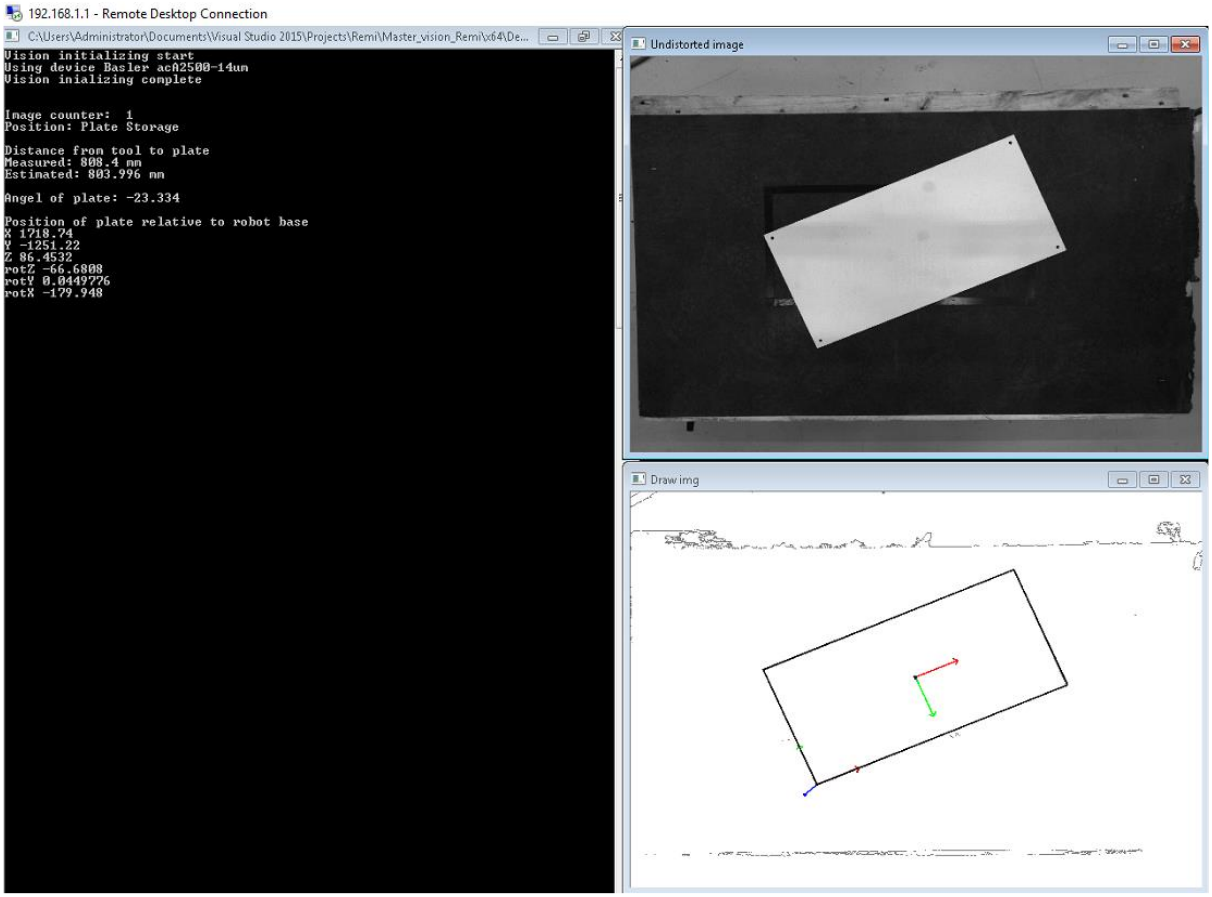
Figur 3.16 HMI i bruk

HMI tilknyttet maskinsynet

HMI tilknyttet maskinsynet er terminal utskriften fra C++ programmet som kjører maskinsynet. Operatøren kan ikke gi kommandoer, dette er bare for informasjon. Bilder fra maskinsynet kan ikke vises i HMIen tilknyttet PLSen. På grunn av dette ble det valgt å ha to separate HMIer. Figur 3.17 vis en todelt skjerm der til venstre vises terminalutskriften og til høyre vises to bilder tatt under maskinsyn prosessen.

Terminalutskriften viser følgende:

- Bildeteller.
- Posisjon – Om roboten er ved platelager eller vegg.
- Distanse fra verktøy til plate, både estimert og målt med Lidar sensor.
- Vinkel på platen.
- Posisjonen til platen relativ til robotens base.



Figur 3.17 Maskinsyn HMI

Bildene som blir vist er

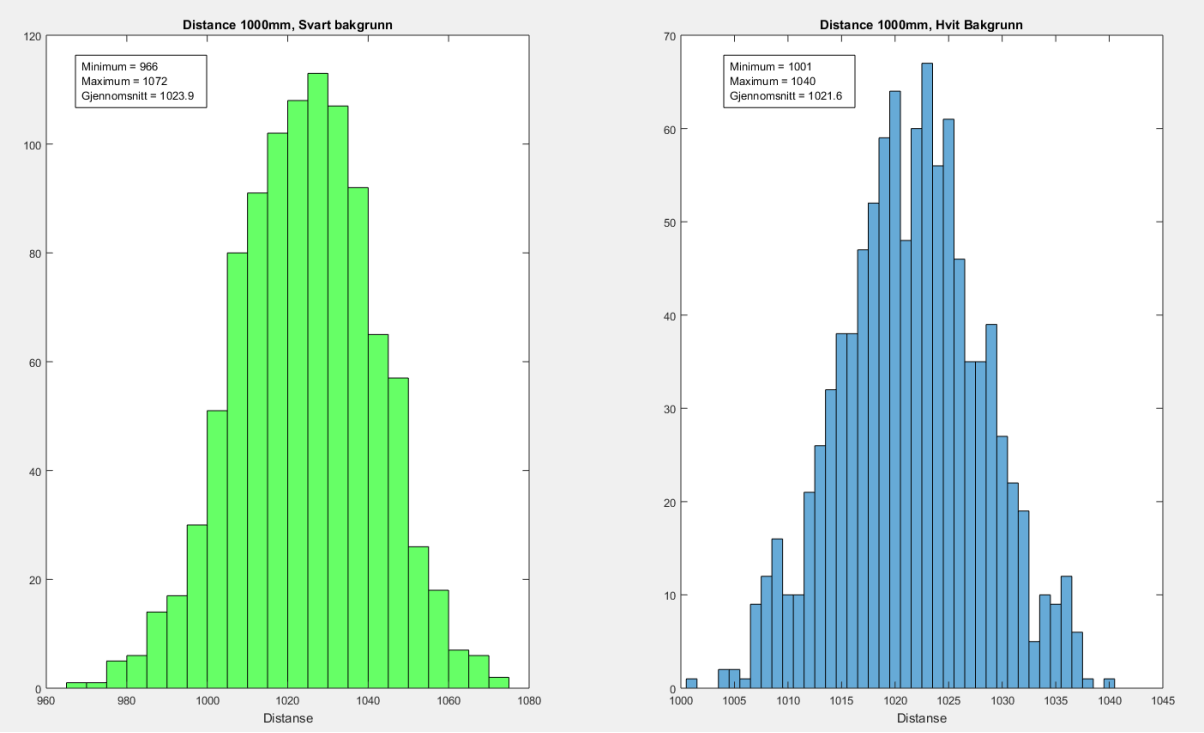
- Bilde tatt av maskinsynet (Uforvrengt bilde)
- Estimert posisjon grafisk vist på platen. Koordinatsystem som kan observeres midt på platen er den estimerte midtposisjonen av platen. Rød pile er X-akse og grønn pil er Y-akse.

3.3.4 Avstandsmåling

Verktøyet kom med Adafruit VL53L0X Lidar sensor [28]. En Lidar måler avstand til et mål ved å lyse opp målet med et pulserende laserlys og måler den reflekterte pulsen. Sensoren bruker en snever lyskilde og er best til å måle avstand direkte foran sensoren. Sensoren kan handtere målinger mellom 50 – 1200mm. Ut i fra omgivende lys og avstand vil sensoren gi en nøyaktighet på 3 til 12% [35].Godt omgivende lys og skinnende overflater vil gi best resultat.

Sensoren kan leses av en mikrokontroller og krever spenning på $3V - 5V$. Siden CX2040 kontrolleren ikke kom med støtte for denne type målinger ble det valgt å implementere en Arduino for avstandsmåling. Arduino kommer ikke med Ethernet tilkobling men ved å montere et Ethernet-kort åpner denne muligheten seg. Arduino kan da kommunikasjon med CX2040 kontrolleren.

En av utfordringene med denne sensoren var nøyaktighet. 3 til 12 % på en meter tilsvarer 30 til 120mm feil. Figur 3.18 viser 1000 distanse målinger gjort med Lidar sensoren. Det observeres at det er mindre differanse når Lidar sensoren måler direkte på den hvite platen framfor den svarte platen

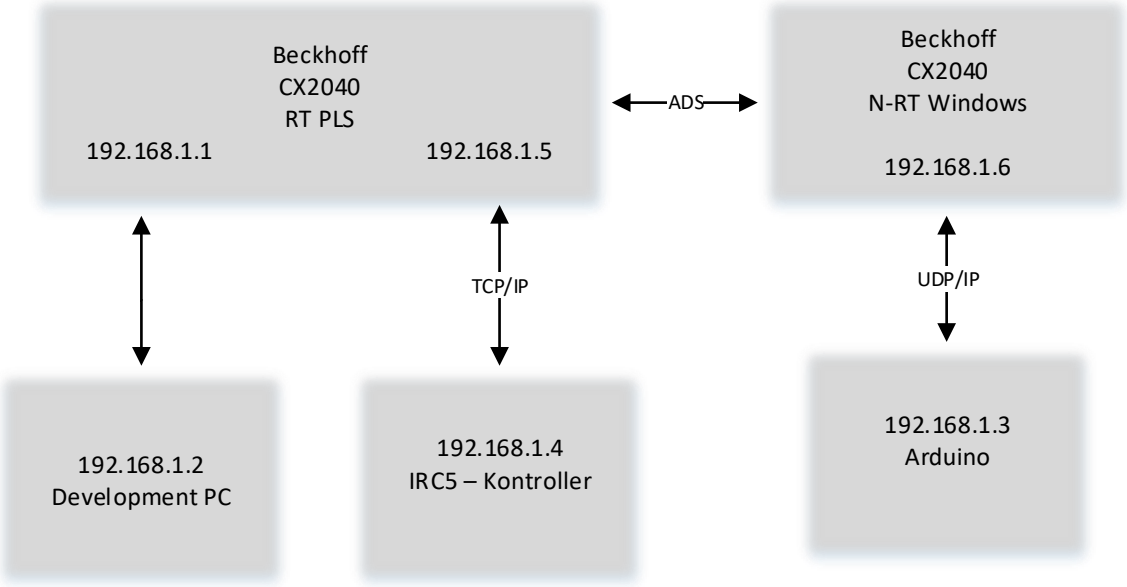


Figur 3.18 1000 målinger med Lidar sensor

På grunn av den store spredningen på målingene ble det valgt å ikke bruke denne under posisjonsestimering. Sensoren sammen med mikrokontrolleren var også noe ustabil da den ikke ville gi målinger i perioder. Sensoren er fortsatt i systemet og brukes til kontroll måling men har ikke en aktiv rolle i kontrollsystemet.

3.4 Kommunikasjon

I dette prosjektet er det forskjellige systemer som må kommunisere for at ønsket funksjonalitet skal oppnås. Figur 3.19 viser hvordan de forskjellige undersystemene kommuniserer sammen. Development PC'en er ikke nødvendig for å kjøre systemet da CX2040'en er satt opp slik at operatøren har tilgang til både PLS og maskinsyn HMI. Det er derimot mulig og monitorere og aktivt samhandle med PLS fra Development PC'en. CX2040 egnes ikke til utvikling av både PLS og N-RT applikasjoner samtidig da den går noe tregt, den egner seg best til å kjøre systemet og feilsøking.



Figur 3.19 System kommunikasjon

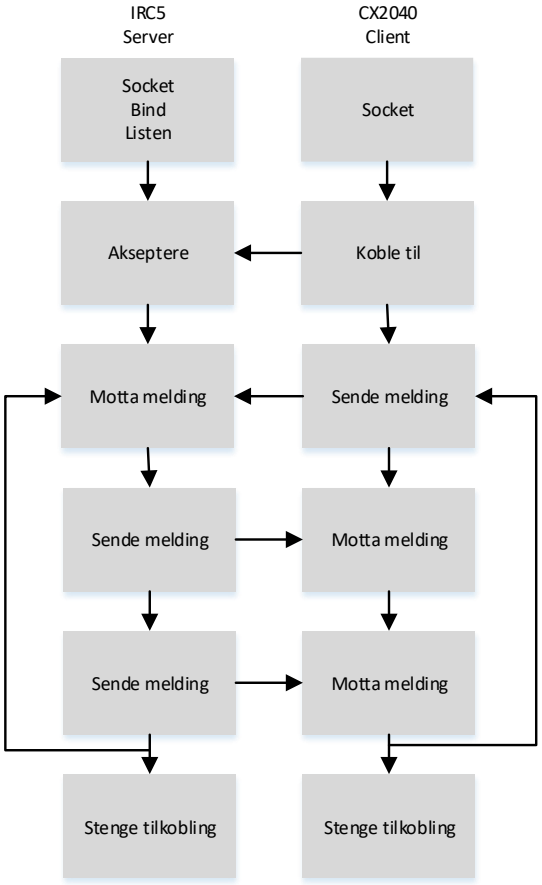
3.4.1 TCP/IP

Kommunikasjonen mellom PLS og IRC5 skjer via TCP/IP. Dette ble valgt da noe av koden til det tidligere prosjektet på denne roboten [28] ble gjenbrukt og der ble det brukt TCP/IP. Det er også en sikrere kommunikasjonsmåte en UDP.

Kommunikasjonen mellom PLS og IRC5 ble satt opp slik at IRC5 er server. Da lager IRC5 tilkoblingen og binder den. Deretter venter den på at noen skal prøve å koble seg til. Når CX2040 prøver å koble seg til vil IRC5 akseptere forbindelsen. Etter forbindelsen er opprettet vil IRC5 venter på en innkommende beskjed, da den bare har en utførende karakteristikk i dette systemet. Kommunikasjonen er satt opp som vist i Figur 3.20 og fungerer på følgende måte når tilkoblingen er etablert.

- IRC5 venter på data.
- CX2040 sender en predefinert streng av bokstaver og tall.
- IRC5 mottar, sjekker at den er definert og sender et ekko tilbake.
- IRC5 utfører handlingen definert.
- IRC5 bekrefter at handling er utført med å sende et nytt ekko.
- IRC5 venter på data (syklusen repeteres).

Og sende en streng av bokstaver og tall fram og tilbake er primitiv bruke av TCP/IP protokollen. Da strengen må deles opp og konverteres til brukbare datatyper i begge kontrollere.

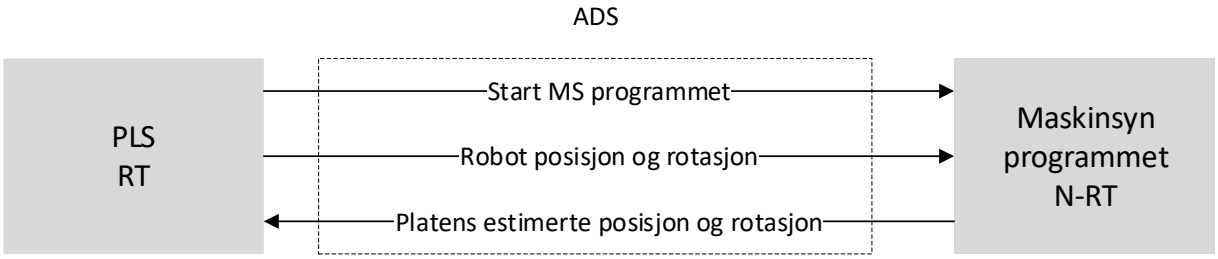


Figur 3.20 IRC5 og CX2040 kommunikasjon

3.4.2 ADS

ADS protokollen brukes mellom PLS og maskinsynet. Figur 3.21 illustrerer i hovedtrekk hvordan ADS kommunikasjonen brukes i dette systemet. PLS bestemmer når maskinsynet starter ved å sette en startvariabel høyt. Maskinsynet starter da en syklus der den tar et bilde og analyserer det. Når maskinsynet skal estimere platens posisjon trenger den robotens nåværende posisjon. Denne posisjonen sendes fra PLS til maskinsynet. Når platens posisjon er estimert sender maskinsynet denne til PLSen og videre handlinger kan utføres.

I dette systemet pakkes ikke variablene som sendes over ADS protokollen. Dette fordi Beckhoff har gjort det mulig å kalle variabler direkte med navn på en effektiv måte. I Figur 3.21 når platens estimerte posisjon og rotasjon sendes tilbake til PLSen blir dette gjort med mange variabler og ikke en.



Figur 3.21 ADS kommunikasjon

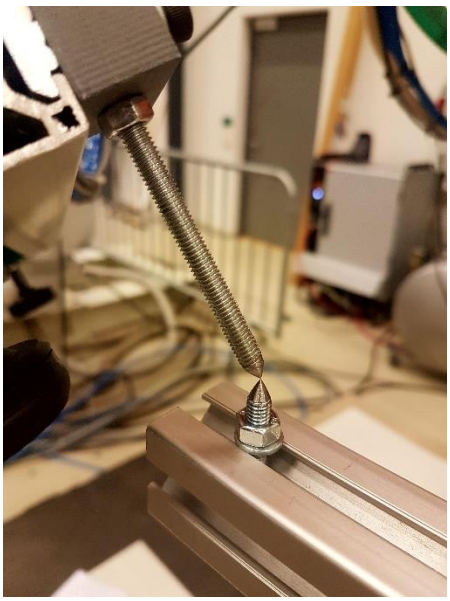
3.5 Kalibrering av robot

I dette kapittelet beskrives metodene bak implementering og kalibrering av ABB IRB6600.

3.5.1 Verktøy kalibrering

I dette prosjektet var det to tilgjengelige metoder for å finne verktøyets senter (VS): 3D-modell eller manuell kalibrering. Siden nøyaktigheten mellom 3D-modell og virkelige verktøyet var ukjent, og den relativt gode nøyaktigheten som kan oppnås med manuell kalibrering ble denne metoden valgt.

Manuell kalibrering er avhengig av operatørens ferdighet og øyemål for å treffe det kjente objektet. For å utføre kalibrering ble det 3D-printet en holder for en M6 bolt som kunne monteres på aluminiumsskinnene på verktøyet. M6 bolten ble slipt ned til en spiss som vist i Figur 3.22. Ved å bruke en lang bolt ga dette justeringsmuligheter for plassering. En slipt M6 bolt ble også brukt som referansepunkt under kalibrering.



Figur 3.22 Verktøy kalibrering

VS defineres med fire punkter [23]. Det er også mulig å definere koordinatsystemets retning når verktøyet kalibreres. Under kalibrering kan følgende metode defineres:

- TCP (Default orient.) – setter orientering av TCP lik som koordinatsystemet ved robotens base.
- TCP&Z – Setter orienteringen i ønsket Z retning.
- TCP&Z,X – Setter orientering i ønsket X og Z retning.

Når VS skal kalibreres er det fordelaktig og plassere kalibreringsverktøyet der ønsket VS skal være. Det var ikke mulig å plassere kalibrerings verktøy i midten av verktøyet grunnet ventilblokkene til vakuum systemet. Dette ble løst ved å plassere kalibreringsverktøyet utenfor ønsker VS også flytte det i ettertid.

3.5.2 Øye-hånd kalibrering

For å finne distanse og rotasjon, X, av kamera i forhold til VS utføres et øye-hånd kalibrering som beskrevet i 2.4.3. Sondre Tørdal har laget en MATLAB funksjon [36] som løser ligning 9 fra kapittel 2.4.3. Distanse og rotasjon mellom robot base og VS, H_t, kan hentes ut fra IRC5 kontrolleren. Distanse og rotasjon mellom kamera og kalibreringsmønster H_m hentes ut fra OpenCV.

For å få et tilfredsstillende resultat er det nødvendig med mange målepunkter og i tilfeldig rekkefølge. Kalibreringen ble automatisert til en viss grad. Et C++ program ble laget som kommuniserte med IRC5 kontrolleren og hentet ut H_t. Programmet tok også et bilde av kalibreringsmønsteret, analyserte dette og hentet ut H_m. Figur 3.23 viser kalibreringsmønsteret der algoritmene har funnet hjørnene og merket disse. Algoritmen har også funnet origo i mønsteret og merket dette med et koordinatsystem.

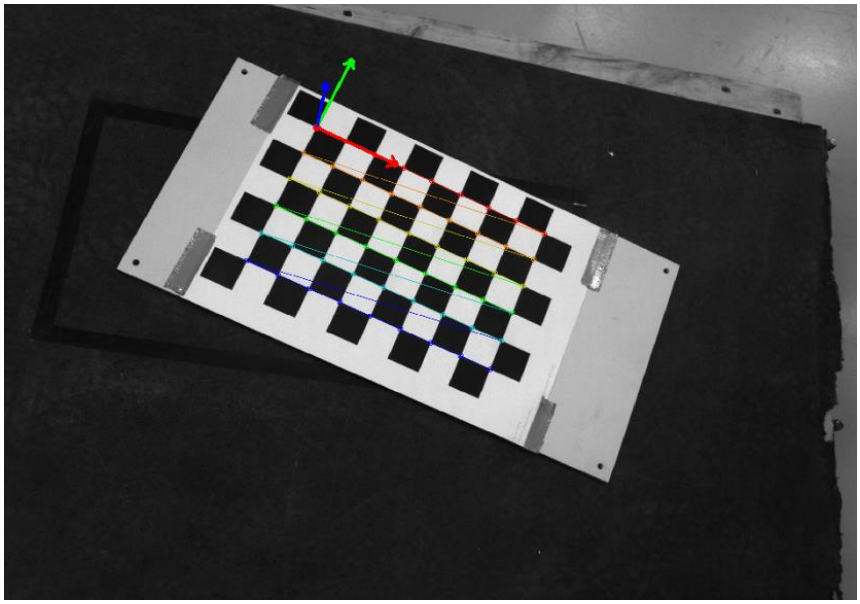
Kalibreringen fungert som følger:

C++ programmet

- Flytte roboten manuelt til ønsket posisjon.
- Trykk Enter for å kalkulere H_t og H_m.
- Lagrer automatisk H_t og H_m tile et .txt dokument.
- Repeter prosessen.

MATLAB programmet:

- Leser .txt filen og henter ut H_t og H_m.
- Kalkulerer $A_i \; og \; B_i$.
- Kalkulerer X som er distanse og rotasjon mellom VS og kamera.



Figur 3.23 Øye-hånd kalibrering

4 Resultat

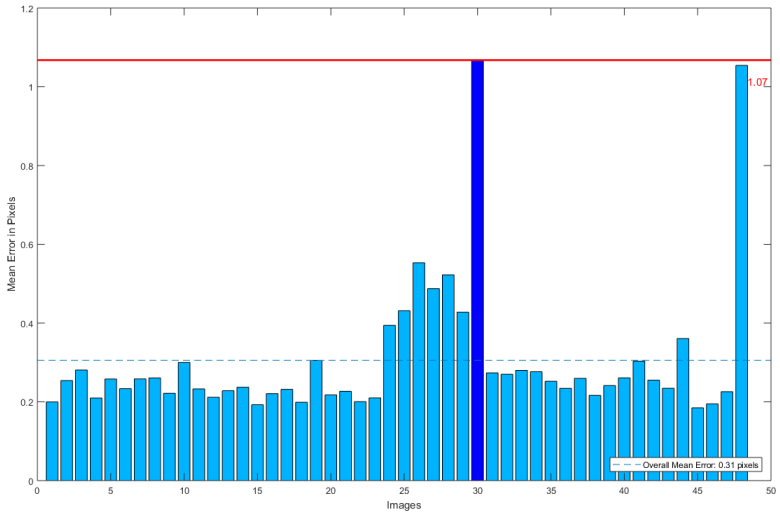
4.1 Kalibrering

Kalibrering av de forskjellige delene av systemet er en vital del for å oppnå ønsket virkemåte og god nøyaktighet og repeterbarhet. Kalibrering har vært en utfordring og i dette kapittelet vil resultatene fra kalibreringen vises og diskuteres.

4.1.1 Kamera kalibrering

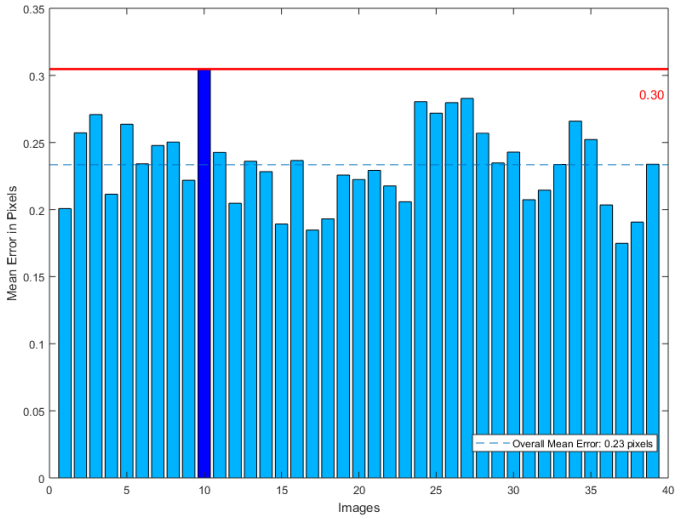
Kamera kalibrering ble utført flere ganger for å få best mulig resultat. For å få et tilstrekkelig resultat er det viktig å ta bilde over hele kameraets synsfelt. Det er også viktig å ta bildene av kalibreringsmønsteret fra forskjellige avstander og fra forskjellige vinkler.

Kalibrering ble utført med 48 bilder tatt fra forskjellige vinkler og avstander. Figur 4.1 viser gjennomsnittlig reprojeksjonsfeilene til hvert enkelt bilde. Ekstremverdier på over en piksel kan observeres. Disse ekstremverdiene kan forårsakes av bildestøy, feil på kamera eller objektoverflaten, ujevn belysning eller feil ved detektering av kalibreringsmønsteret [37].



Figur 4.1 Reprojeksjons feil

I denne avhandlingen vil bilder som forårsaker ekstremverdier bli fjernet. Kalibreringen ble kjørt på nytt etter disse ble fjernet. Figur 4.2 viser resultatet etter korrigering. Gjennomsnittlig reprojeksjonsfeil etter korrigering er på 0.23 piksler.



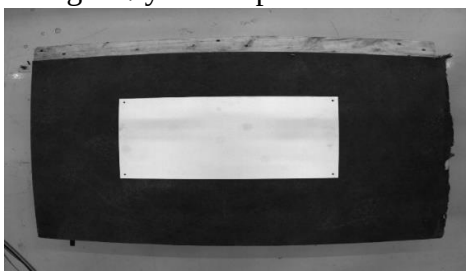
Figur 4.2 Reprojeksjons feil etter korrigering

Kalibreringen gav resultatene presentert i Tabell 4.1. Disse brukes for å korrigere bilde og resultatet kan observeres i Figur 4.3 som viser det originale bilde, og Figur 4.4 som viser det korrigerte bilde. Fokal lengden oppgitt er 4 mm og kalibreringen estimererte noe høyere verdi, dette er fordi justering av fokus endrer litt på fokal lengden. Senterpunktet uti fra kameras oppløsning er $1296x972$ mens estimert er $1299.5x1025.5$. I x- og y-retning er senter forskjøvet med 3.5 og 53.5 piksler.

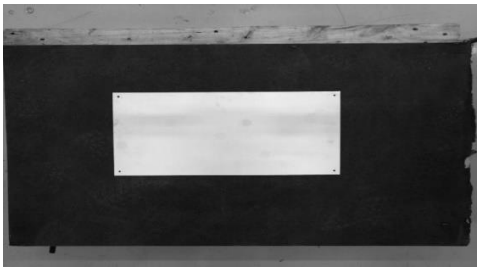
Kamera og linse kalibrering		
Fokal lengde	f_x	$4.078\ mm$
	f_y	$4.084\ mm$
Senterpunktet	c_x	$1299.5\ piksl$
	c_y	$1025.5\ piksl$
Radiell forvrenging koeffisient	k_1	-0.2608
	k_2	0.1386
	k_3	-0.0425
Tangentiell forvrengings koeffisient	p_1	$3.412 \cdot 10^{-5}$
	p_2	$1.0446 \cdot 10^{-4}$

Tabell 4.1 Resultater fra kamera kalibrering

Under kalibrering ble bildene tatt ved å manuelt jogge roboten til ønskede posisjoner. Dette er en tidkrevende prosess som kunne vært forbedret ved å lage et program som automatisk flytter roboten rundt og tar bilder. Kalibrering av kamera og linse kunne også vært kombinert med kalibrering av øye-hånd parameter.



Figur 4.3 Original bilde



Figur 4.4 Korrigert bilde

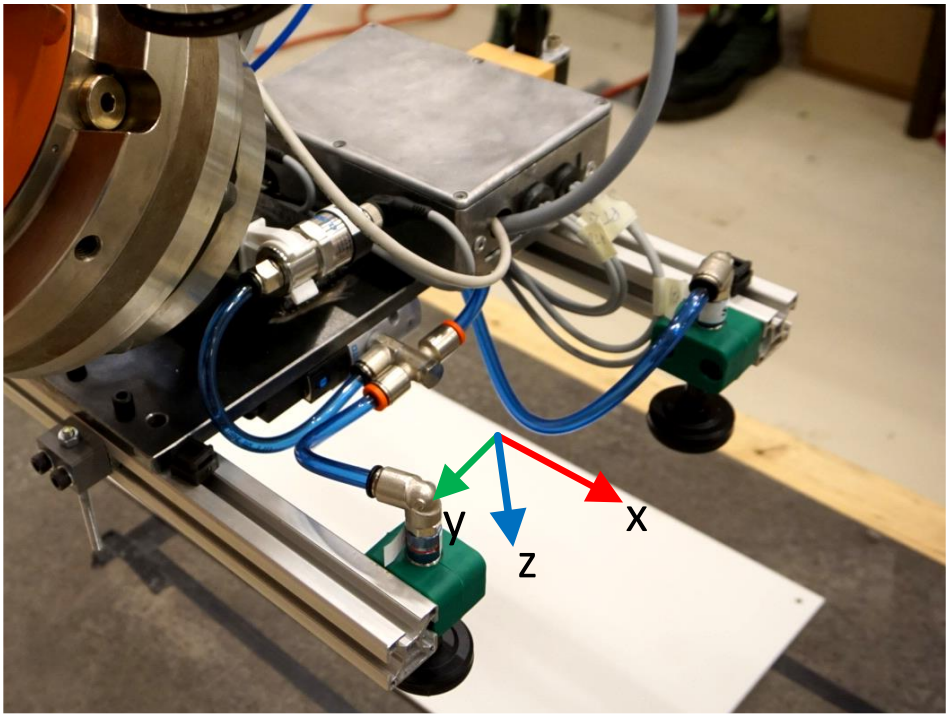
4.1.2 Verktøy kalibrering

Kalibrering av VS var en av de mindre tidkrevende kalibreringene, men tok noen forsøk. Etter gjentatte forsøk ble resultatene av kalibreringen akseptabel og disse er presentert i Tabell 4.2. Gjennomsnittsfeil på 0.19 mm ble oppnådd etter flere forsøk og under 0.5 mm er sett på som akseptabelt. Først ble verktøyet grovkalibrert for å få et VS som kunne brukes når kalibreringen ble gjentatt.

VS kalibrering	
Max Error	0.2523 mm
Min Error	0.0160 mm
Gj.snitt Error	0.1958 mm
X	168.4338mm
Y	7.139796 mm
Z	159.5777mm

Tabell 4.2 Resultater av verktøysenter kalibrering

Orienteringen var litt vanskeligere å få korrekt ved manuell kalibrering. Dette ble løst ved å gjenbruke orienteringen fra den tidligere gruppen da denne stemte godt. Orienteringen ble rotert for å få samme orientering som kameraets koordinatsystem. Orienteringen er vist i Figur 4.5. X retning går langs de tverrgående aluminiumsskinnene. Y går langs senter av verktøyet og positiv Z retning går ut av verktøyet ned mot platen. Kalibreringsverktøyet kan observeres i bak kant av verktøyet og er montert på motsatt side i forhold til kamera. Spissen er montert slik at den er på samme nivå som bunnen av sugekoppene. Dette gjør at kalibrert VS bare må forflyttes langs Y akse for å finne ønskede VS. Dette ble gjort i RobotStudio.



Figur 4.5 Verktøyets koordinatsystem

4.1.3 Øye-hånd kalibrering

For å finne distansen og rotasjonen mellom kamera og VS ble det gjennomført øye-hånd kalibrering og en manuell måling. Kalibreringen var noe tidkrevende da roboten måtte manuelt jogges til forskjellige posisjoner. For å få et tilfredsstillende resultat er ca 60 målinger nødvendig [26]. Det ble gjennomført 30 målinger på grunn av tidsbegrensinger. Resultatene kan observeres i Tabell 4.3. Manuelle målinger og justeringer av verdiene gav bedre resultat enn kalibreringen. Dette ble testet ved å plassere verktøyet i senter av platen ved bruk av maskinsynet.

Kalibreringen og de manuelle målingene viser at det er en del forskjell mellom X-retning og Z-retning. Flere målinger er nok nødvendig for et mer tilfredsstillende resultat. Tiden kalibreringen tok var en begrensende faktor og prosessen kunne vært automatisert.

	Manuell [mm]	10 målinger [mm]	20 målinger [mm]	30 målinger [mm]
X	0	10.0	2.4	5.1
Y	-160.2	-191.5	-200.1	-196.8
Z	-30	-30.2	-27.1	-29.15

Tabell 4.3 Resultat av øye-hånd kalibrering

4.2 Maskinsyn

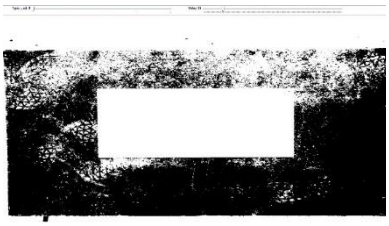
Her presenteres resultatene fra maskinsynet og posisjonsestimering. Hvordan dette ble testet vil også bli beskrevet.

4.2.1 Bildebehandling

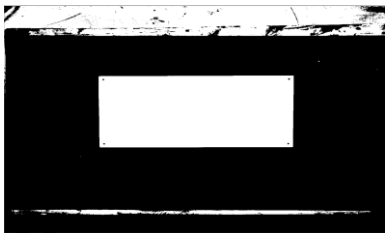
Metoden som brukes for å gjøre bilde binært har mye å si for hvor lang tid det tar å finne konturene i bilde. Som beskrevet tidligere er det to forskjellige forhold når platen skal plukkes/plasseres med forskjellige bakgrunner og lysforhold.

Platelager:

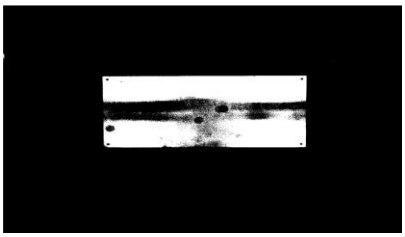
Ved platelageret stopper roboten i samme posisjon hver gang for å ta bilde av hele pallen. En enkel grenseverdi operasjon blir brukt. Dette ga et stabilt og rask bildebehandling. Flere forskjellige verdier ble testet for å finne optimal verdi. I Figur 4.6 er grenseverdien satt til 35, og konturalgoritmen ser etter sammenhengene hvite piksler. Dette er ikke en tilfredsstillende verdi da prosessen videre går tregt og det er vanskelig å skille plate fra bakgrunnen. Figur 4.7 viser grenseverdi på 120 og det er tydelig hvor platen starter og stopper. Høyere verdier fører til at platen begynner å blende inn i den svarte bakgrunnen igjen som vist Figur 4.8 hvor midten av platen begynner å forsvinne.



Figur 4.6 Platelager, grenseverdi 35



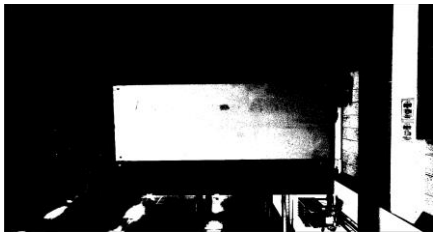
Figur 4.7 Platelager, grenseverdi 120



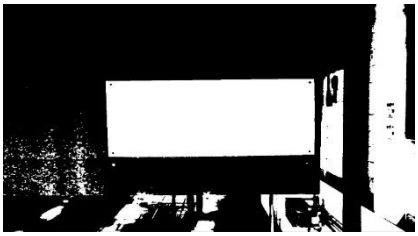
Figur 4.8 Platelager grenseverdi 190

Vegg:

I dette tilfellet var det nødvendig med andre metoder enn vanlig grenseverdi, da det var mye lysere bakgrunn og lettere for at bakgrunn blendet inn med bakgrunnen. Figur 4.9 viser vanlig grenseverdi operasjon og platen begynner å forsvinne ved grenseverdier lavere enn ved platelageret. Ved å bruke et slørings filter og OTSU grenseverdi metoden, vist i Figur 4.10, ble det tydelig hvor platen starter og slutter. Denne metoden førte til at maskinsynet stabilt og raskt fant platen.



Figur 4.9 Vegg, grenseverdi 100



Figur 4.10 Vegg, OTSU grenseverdi metode

Maskinsynets arbeidsflyt når roboten er ved veggen er vist Figur 4.11. Arbeidsflyten ved platelagere veldig lik, eneste som endrer seg er som nevnt over, grenseverdioperasjonen.

1. Kamera tar et svart/hvit bilde med maks oppløsning, 2592piksler x 1944 piksler.

2. Forvrenging i bilde rettes opp.

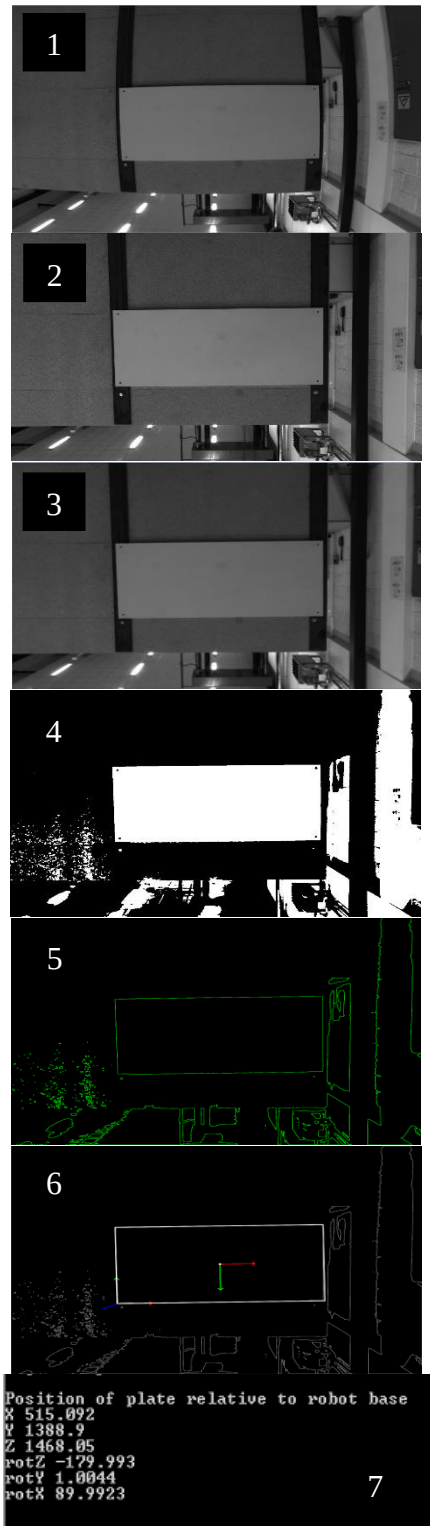
3. Et slør filter blir tilført bilde.

4. OTSU grenseverdi algoritmen kjøres for å finne optimal grenseverdi, dette gjør bilde om til binære verdier.

5. Kontur algoritmen brukes for å finne alle konturene i bilde. De grønne strekene er konturene funnet

6. `cv::RotatedRect` algoritmen finner rektangelet i bilde og returnerer de 4 hjørnene til platen. Siden dimensjonen til platen er kjent brukes `cv::SolvePnP` algoritmen til å estimere platens posisjon iforhold til kamera.

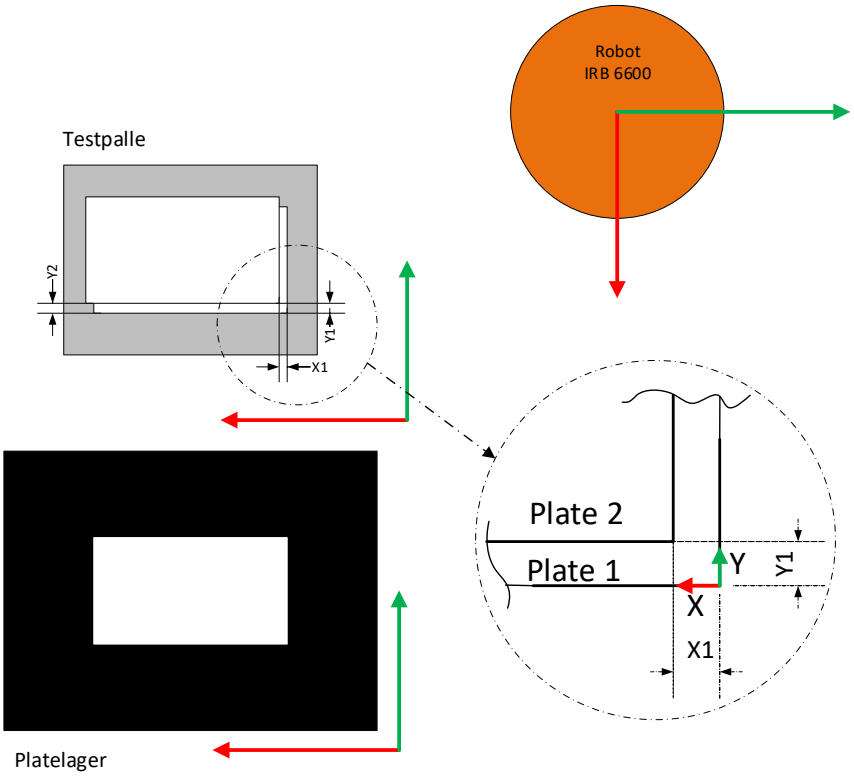
7. Platens posisjon relativ til robotens base kalkuleres med kinematikken forklart i 3.2.4. Platens posisjon sendes til PLS.



Figur 4.11 Maskinsyn arbeidsflyt

4.2.2 Repeterbare av maskinsyn

For å teste maskinsynets nøyaktighet og repeterbarhet ble det satt opp noen tester. De første fire testene ble gjennomført ved å plukke opp plater ved platelageret og flytte de til testpallen. Platene plasseres på samme punkt på testpallen hver gang. Figur 4.12 viser oppsettet. For å måle hvordan maskinsynet opptrer under testene ble platene på testpallen målt i forhold til den første platen som blir lagt ned. Denne testen ble brukt for å kunne måle avviket mellom de plasserte platene. Skyvelære ble brukt for å utføre de fysiske målingene.



Figur 4.12 Testoppsett 1

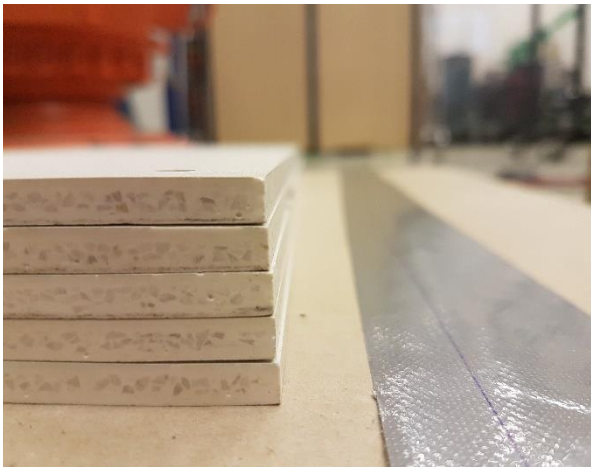
De fire testene ble gjennomført følgende måte:

Test

1. 5 plater stablet midt på platelagere uten vridning flyttes med roboten til et fast punkt på testpallen. Testen er kjørt to ganger derav test 1.1 og 1.2.
2. 5 plater stablet midt på platelageret med vridning flyttes med roboten til et fast punkt på testpallen. Testen er kjørt to ganger derav test 2.1 og 2.2.
3. 5 plater lagt på pallen uten vridning, en etter en, en på midten og en i hvert hjørne. Testen er kjørt to ganger derav test 3.1 og 3.2.
4. 5 plater lagt på pallen med vridning, en etter en, en på midten og en i hvert hjørne. Testen er kjørt to ganger derav test 4.1 og 4.2.

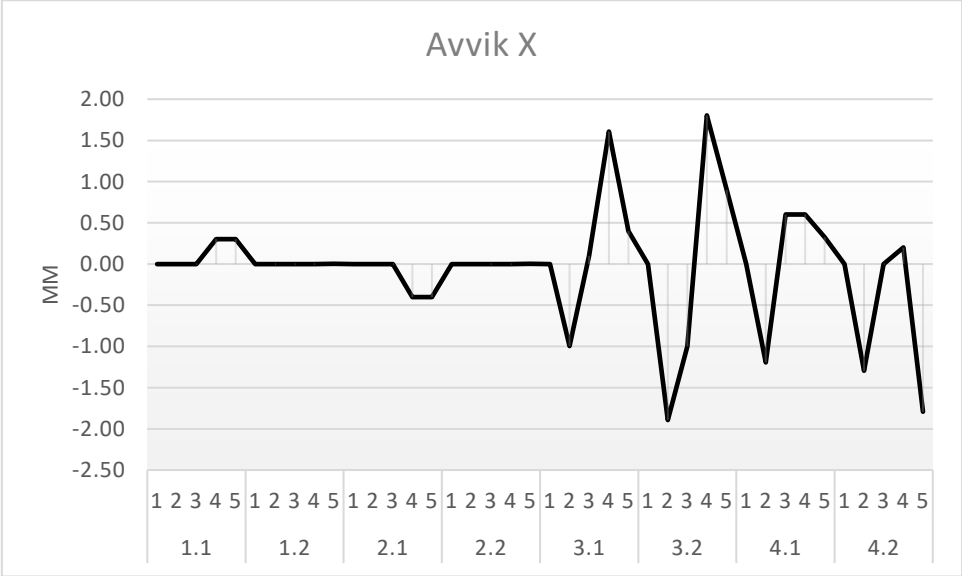
Resultater test 1-4

I Figur 4.12 viser X-retning langs den røde pilen og Y-retning langs den grønne. Avstandene X_1, Y_1, Y_2 blir målt fra en plate til neste. Disse blir så konvertert slik at avstanden er i forhold til den første platen lagt. X_1, Y_1, Y_2 brukes så til å regne ut platens senter punkt og vinkel. Avviket mellom platenes senter blir så plottet. De første testene viser at repeterbarheten er relativt høy nær senter av kamera, dette kan observeres i Figur 4.13 som viser et bilde fra test 1.1. Når platene ligger utenfor senter av kamera går repeterbarheten ned og det blir større avvik i både X og Y retning, dette er tydelig fra Figur 4.14 som viser et bilde fra test 3.1 Figur 4.15 viser avviket i X fra de fire første testene og igjen det kan observeres at ved test 1 og 2 er det relativ lite avvik men den øker ved test 3 og 4. Figur 4.16 viser avviket i Y-retning.

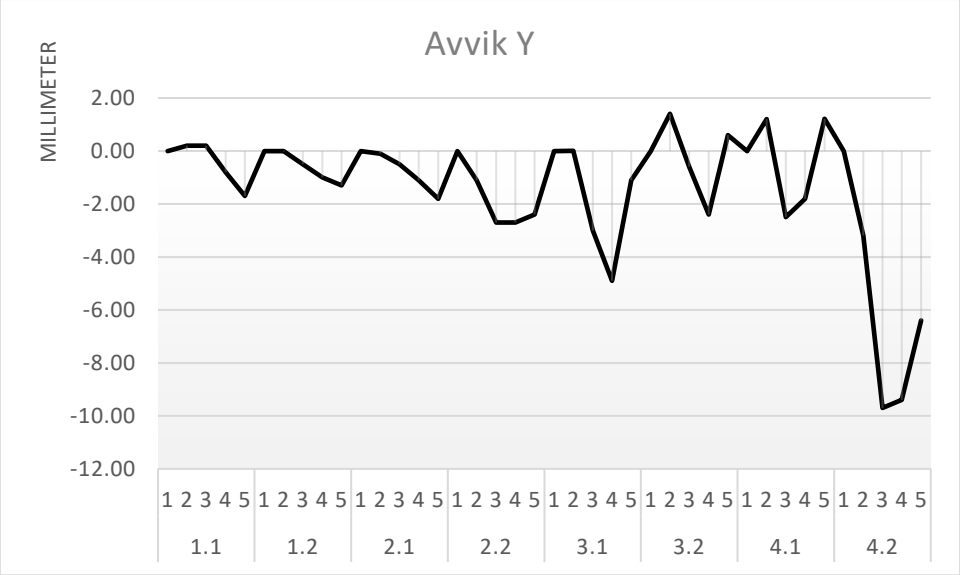


Figur 4.13 Illustrasjon av resultat ved test 1.1

Figur 4.14 Illustrasjon av resultat ved test 3.1

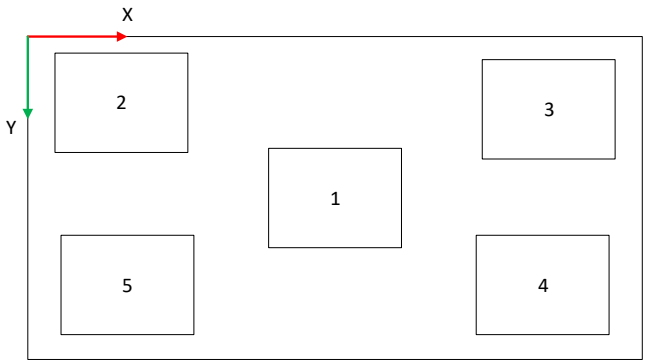


Figur 4.15 Avvik X ved test 1 til 4



Figur 4.16 Avvik Y ved test 1 til 4

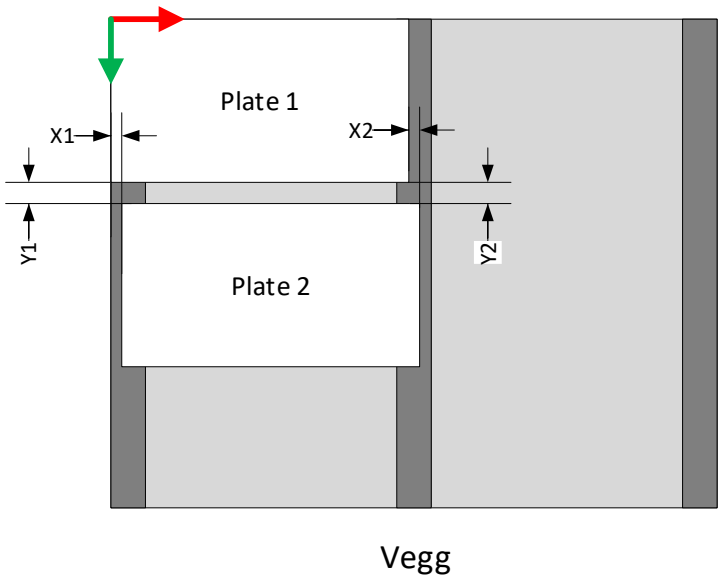
Det er en del usikkerhet ved test 1-4 da platene flytter seg litt når neste plate blir løftet av ved platelageret og når platene blir stablet oppå hverandre på testpallen. Det er også et usikkerhetsmoment ved å måle med skyvelære, men samme personen utfører målingene hver gang for å minske målefeil. Metoden med å måle ut ifra nederste platen kan gi følgefeil. Testene gir derimot en indikasjon på hvor eventuelle feilkilder kan være. En mulig grunn til at feilen blir større når platen flyttes ut fra senter av platen kan være den store forvrengingen i linsa og ikke tilstrekkelig kalibrering.



Figur 4.17 Platelager sektorer

Figur 4.17 viser hvordan platene kan ligge på platelageret ved test 4. Det ble observert under testing at maskinsynet estimerte korrekt distanse fra kamera til platen når platen lå i sektor 1,2 og 4. Når platen lå i sektor 5 estimerer maskinsynet for kort distanse og i sektor 3 estimeres for lang distanse.

For å gi en visuell beskrivelse av systemet ble det kjørt noen tester der roboten plukker opp platen fra platelageret og plasserer den på veggen. Da bruker roboten maskinsyn til både plukke og plasseringsoperasjonen. Figur 4.18 viser hvordan platene plasseres på veggen. Plate 1 henger på veggen, den henger på samme plass hele tiden, kamera tar bilde av plate 1 og plaserer da plate 2 i forhold til plate 1. Når det blir tatt bilde av plate 1 er den i senter av kamera, fordi fra foregående tester ble det vist at systemet blir mindre nøyaktig utenfor senter. Platene måles i forhold til hverandre med skyvelære.

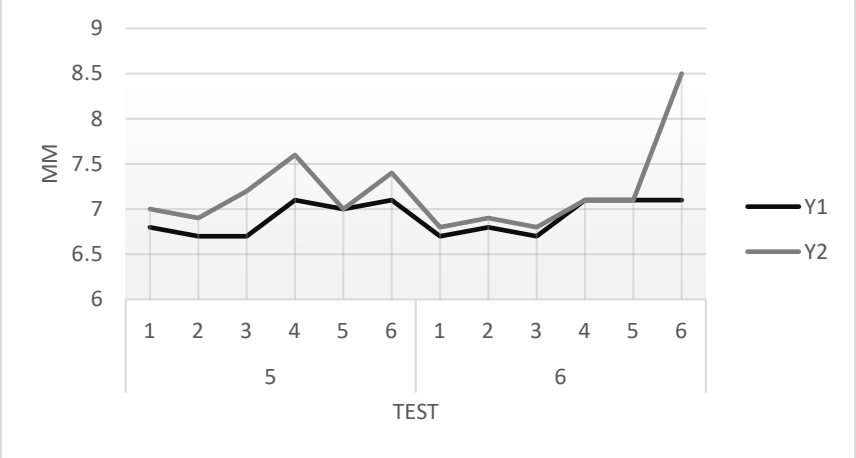


Figur 4.18 Testoppsett 2

Systemet er lagt opp slik at systemet kan plassere en plate under den som er fast plassert. Så den samme operasjonen blir gjentatt flere ganger. Følgende tester ble utført når platen skulle plasseres på vegg.

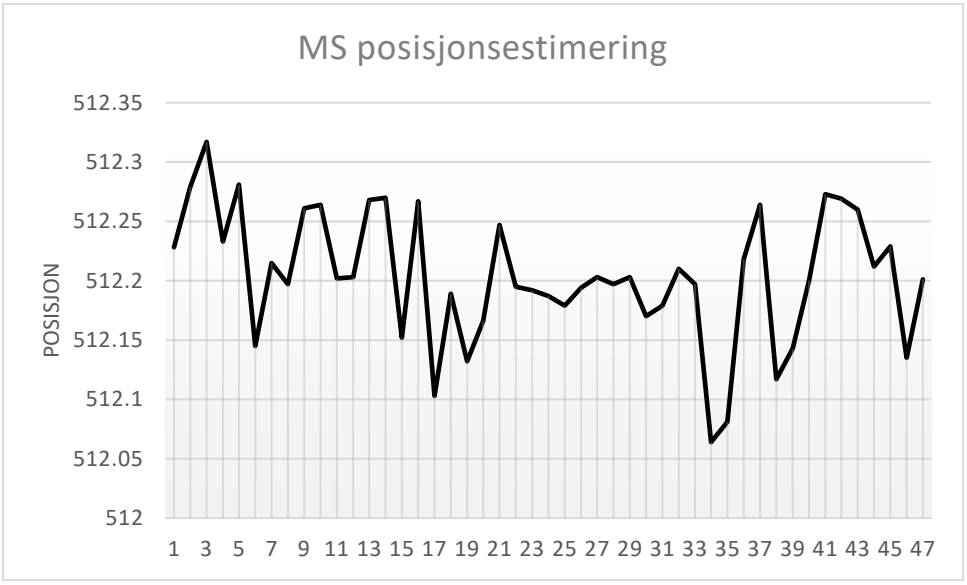
5. Plate ligger midt på platelagere uten vridning og blir plassert på veggen
6. Platen ligger midt på platelagere med vridning og blir plassert på veggen.

Figur 4.19 viser distansen mellom platen målt på to punkter Y_1 og Y_2. Målingene viser at distanse mellom platene ligge rundt 7mm med maks distanse mellom de to punktene på 1.5mm. Test 5 og 6 viser potensialet i maskinsynet men flere tester er nødvendig.



Figur 4.19 Distanse mellom platene ved test 5 og 6

For å teste repeterbarheten til maskinsynet ble det tatt nesten 50 bilder av en plate montert på veggen fra en fast posisjon. Figur 4.20 viser at det er maksimum 0.3 mm variasjon, denne grafen viser estimert X-posisjon av platen. Y-posisjonens variasjonen er også 0.3 mm. Estimert distanse til platen har en variasjon på 1mm. Estimerte vinkler har en variasjon på 0.1 grader.



Figur 4.20 Estimert X-posisjon ved vegg

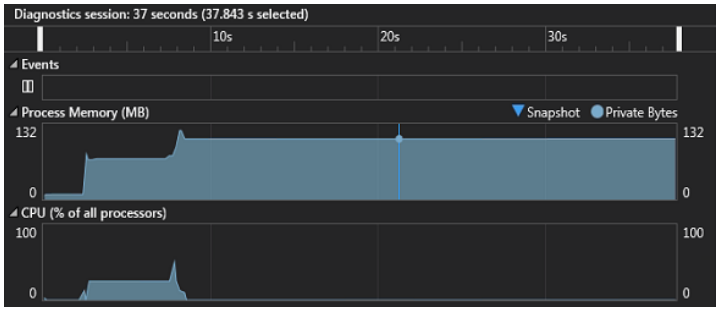
4.2.3 Tidsforbruk og ressursforbruk

For å gi en oversikt over de krevende prosessene i maskinsynet ble prosessene målt. Tabell 4.4 beskriver tiden til de fire hovedoppgavene. Tiden brukt for å ta et bilde og konvertere det til OpenCV er likt ved vegg og platelager. Dette gjelder også for fjerning av forvrenging som er den mest tidkrevende operasjonen. Om oppløsningen nedskaleres kan nok dette føre til mindre tidsbruk men dette vil igjen gå på bekostning av fordelene ved høy oppløsning. Bildeprosesseringen er grenseverdi- og kontur operasjonene. Denne prosessen varierer litt ut i fra om bilde er tatt ved vegg eller platelager, dette er fordi det brukes forskjellige grenseverdi operasjon som krever forskjellig mengde prosessering. Posisjonsestimeringen er det som er minst tidkrevende å gjennomføre.

	Tid ved platelager	Tid ved vegg
Ta et bilde og konvertere til OpenCV	0.6-0.7sek	0.6-0.7sek
Fjerne forvrenging	4.3-4.7sek	4.3-4.7sek
Bildeprosessering	0.1-0.2sek	0.4-0.7sek
Posisjonsestimering	0.1sek	0.1sek

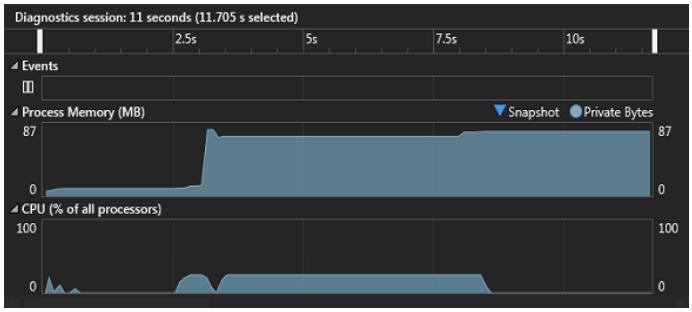
Tabell 4.4 Tidsbruk maskinsyn

Ressursforbruk kan overvåkes i Visual Studio. Figur 4.21 viser ressursforbruket ved en syklus. Prosessminne og brukt CPU kraft i % kan observeres. CPU kraft er oppe i 50% på et punkt av prosessen.



Figur 4.21 Ressursforbruk maskinsyn

Ved fjerne bildene som vises i HMIen kan nødvendig CPU kraft reduseres. Figur 4.22 viser ressursforbruket bare ved å gjøre denne forbedringen og nødvendig CPU kraft er redusert til 25%.



Figur 4.22 Ressursforbruk ved optimalisering

Verdiene vist over er uten å optimalisere koden. I Visual Studio kan programmer kjøres i feilsøkingsmodus (engelsk: debug mode) eller utgivelsesmodus (engelsk: release mode). Testing av dette systemet er gjort i feilsøkinsgsmodus og dette er fordi Visual Studio har en rekke verktøy som gjør feilsøking enklere. Utgivelsesmodus gir muligheter for å optimalisere den utviklede koden.

5 Diskusjon

Denne avhandlingen har vist hvordan Beckhoff CX2040 og OpenCV kan implementeres i et robotsystem. Detektering av et kjent objekt og posisjonsestimering er nøkkelegenskapene til systemet. Dette har blitt illustrert ved å finne en fasadeplate med en tilfeldig posisjon innenfor et avgrenset område og montere den på en vegg ved bruk av en industriell robotarm.

Maskinsynet brukte et Basler aAC2500 kamera med en 4 mm linse. Den høye oppløsningen på 5 Mp gav ett tilfredsstillende piksel/mm forhold som virket til å fungere bra til oppgaven. Den valgte linsen er på den billige siden av skalaen og har derfor mye geometrisk forvrengning. Det ble forsøkt å kalibrere kamera og linse for å motvirke dette. Dette viste seg å være vanskeligere enn antatt da maskinsynet har vanskeligheter med å estimere dybde ved enkelte sektorer i synsfeltet. En linse med mindre geometrisk forvrenging burde testes for å se om dette løser problemet.

OpenCV er et velutprøvd bibliotek. Implementeringen i dette systemet viste seg å være overkommelig selv med lite tidligere programmeringskunnskaper. Det faktum at OpenCV er åpen kildekode, og er veldokumentert bidrar til den enkle implementasjonen. Dersom OpenCV skal brukes er det viktig at systemutvikler er noe kritisk til dokumentasjonen også. Biblioteket er åpen kildekode noe som betyr at deler er skrevet av programvareutviklere og andre deler av entusiaster.

Bildebehandlingen gjennomføres med algoritmer fra OpenCV. Grenseverdioperasjoner og grensesporings metoder er hoveddelen av bildebehandlingen. Metodene brukt i dette prosjektet for lokalisering av ønskede konturer fungerer bra under forholdene gitt i laben. Det er usikkert hvor robust systemet er dersom forhold som lys og bakgrunnsfarger endres, dette er et naturlig steg for videre testing.

Posisjonsestimering ble gjort ved å bruke kjente dimensjoner i bilde. Algoritmene for denne operasjonen er tilgjengelig i OpenCV biblioteket. Posisjonsestimeringen finner posisjon og rotasjon ut fra fire kjente punkt, disse er fire av platens hjørner. Estimeringen blir gjennomført ut fra informasjon gitt fra et enkelt bilde. Dette viste seg å være problematisk på grunn av forvrengingen i linsen, som nevnt tidligere. En mulig løsning er å designe et kontrollsystem med kontinuerlig tilbakemelding framfor en enkel tilbakemelding. Kontrollsystemet kan da plassere robotarmen slik at kamera alltid er i senter av platen der nøyaktigheten er høyest.

Maskinsyn er prosesskrevende og valget av et 5 Mp kamera gjorde det ikke mindre krevende. Et ønske var å se hvordan Beckhoff CX2040 kontrolleren taklet maskinsyn prosesseringen og var stabil under aller operasjoner. Det som var krevende for kontrolleren var å fjerne forvrenginger fra bilde, noe som tok nesten fem sekund. Denne prosessen kan nok bli unngått med en linse med mindre forvrenging. Det hadde vært interessant å implementer et RT

kontrollsystem basert på maskinsyn og observert hvordan CX2040 kontrolleren hadde taklet dette.

C++ språket brukt til maskinsynet er et kraftfullt verktøy med mulighet for objektorientert programmering. I denne avhandlingen er dette ikke utnyttet, og maskinsyn programmet er noe langt og ustrukturert. Dette gjelder også PLS programmet, dette kunne også vært delt opp i flere blokker for å gi et mer oversiktlig og strukturert program.

Systemet består av et verktøy for håndtering av fasadeplatene. Verktøyet bruker sugekopper og vakuum til dette. Trykket blir overvåket av to trykk sensorer og styrt av to magnetventiler. Dette er implementert i CX2040 kontrolleren. Beckhoff har et stort utvalg av I/O moduler som gjør implementering skalerbart ut fra systemets størrelse. TwinCAT programvaren gjør også utvikling av systemer mer strømlinjeformet med sitt oversiktlige og brukervennlige grensesnitt. Dokumentasjonen til Beckhoffs produktporteføljе er enkelt tilgjengelig på nett.

Kommunikasjonen mellom CX2040 og IRC5 ble gjennomført ved å sende en streng med bokstaver. Dette er noe primitivt og tungvint men fungerte til bruken her. Måten IRC5 og Rapid koden er bygget opp er også ikke ideelt. Skal roboten flyttes til nye posisjoner må dette programmeres i både IRC5 og CX2040 kontrolleren.

HMI til systemet har et minimalistisk design og er sparsom på informasjon. Det fungerer bra her i avhandlingen når utvikler også er operatør. Dersom flere skal bruke HMIen kunne dette vært forbedret. En HMI designet i N-RT delen med både robotstyring og maskinsyn informasjon i samme kunne vært et steg videre. Åpenkildekode programvare QT kunne vært implementert for å gi et mer moderne design enn HMIen som er implementert i dag.

Sikkerheten har hele tiden blitt ivaretatt under avhandlings perioden. Under alle operasjoner er systemet kjørt med robot i manuell modus. Dette betyr at operatøren må holde inne en bryter på "Flexpendanten" under kjøring. Slipper operatøren bryteren stopper roboten umiddelbart. I tillegg er det en sikkerhetskrets som stopper roboten dersom sugekoppene trykkes for langt inn. Flere sikkerhetstiltak som system alarm ved for lite vakuum kunne vært implementert.

Kalibrering var en utfordrende del av denne avhandlingen. Kalibrering av kamera og linse viste seg å ikke være tilstrekkelig over hele kameras synsfelt og trenger mer oppmerksomhet. Øye-hånd kalibreringen hadde også sine utfordringer og gav ikke ønskelige resultater. Parameterne måtte derfor justeres manuelt. Dette kunne muligens vært forbedret med flere målinger.

Gjennom design og byggeperioden har systemet gjennomgått kontinuerlig testing. Det har vært en stor fordel og hele tiden ha tilgang til det fysiske systemet. De avsluttende testene av systemet ble gjennomført og manuelle målinger med skyvelære brukes for å dokumentere resultatene. På grunn av tidspress ble skyvelære målinger valgt da det er enkelt å gjennomføre. Målingene gav gode indikasjoner på hvordan systemet opptrådte. Mer avanserte og nøyaktige målemetoder er et naturlig steg videre for å få mer absolutte indikatorer på systemets ytelse.

6 Konklusjon

Denne avhandlingen har vist at det er mulig å bygge et kontrollsystem basert på maskinsyn ved å bruke OpenCV biblioteket. Prosessen med å lokalisere og montere fasadeplater er simulert i et laboratorium. Kontrollsystemet er utprøvd gjennom gjentatt testing under kontrollerte forhold. Systemets styrker og svakheter er dokumentert under test perioden.

Posisjonsestimering ble gjennomført ved å bruke kjente dimensjoner i bilde og OpenCV algoritmer. Denne metoden ble implementert med gode resultater i senter av bilde og varierende resultater jo lenger objektet beveger seg fra senter. Gjentatte tester har vist at en av de store utfordringene med maskinsyn-basert kontrollsystemer, er kalibreringen involvert. Dersom kalibrering ikke er gjort med tilstrekkelig gode resultater vil det påvirke videre kalibrering og tilslutt det fulførte systemet.

Beckhoff CX2040 kontrolleren fungerte bra som hovedkontroller i systemet og taklet maskinsynet bra. Systemet som er utviklet, har et forbedringspotensial, og da spesielt kalibrering, kommunikasjon og HMI. Systemet har potensial som en plattform for videre utvikling.

7 Referanser

[1] A. K. Knutsen, «Bruker offshoreteknologi i byggeprosjekter,» Forskningsrådet, 14 August 2017. [Internett].
Available:
https://www.forskningsradet.no/no/Artikkel/Bruker_offshoreteknologi_i_byggeprosjekter/1254028353572.
[Funnet Januar 2018].

[2] P. Corke, Robotics, Vision and Control, Springer-Verlag, 2013.

[3] OpenCV, «Camera Calibration and 3D Reconstruction,» November 2017. [Internett].
Available:
https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html#calibrationmatrixvalues.
[Funnet Desember 2017].

[4] Zhengyou Zhang, «A Flexible New Technique for Camera Calibration,»
IEEE transactions on pattern analysis and machine intelligence, VOL. 22,
pp. 1330-1334, 2000.

[5] J. Heikkila og O. Silvn, «A Four-step Camera Calibration Procedure with Implicit Image Correction,» *IEEE International Conference on Computer Vision and Pattern Recognition,* 1997.

[6] OpenCV, «Basic Thresholding Operations,» Desember 2018. [Internett].
Available: https://docs.opencv.org/master/db/d8e/tutorial_threshold.html.
[Funnet Desember 2017].

[7] N. Otsu, «A Threshold Selection Method from Gray-Level Histograms,» *IEEE transactions on systrems, man, and cybernetics,* vol. 9, nr. 1, pp. 62-66, 1979.

[8] S. Suzuki og K. Abe, «Topological structural analysis of digitized binary images by border following,» *Computer Vision, Graphics, and Image Processing 30,* pp. 32-46, 1985.

[9] Wikipedia, «Perspective-n-Point,» 17 September 2017. [Internett].
Available: https://en.wikipedia.org/wiki/Perspective-n-Point.
[Funnet Januar 2018].

[10] OpenCV, «Real Time pose estimation of a textured object,» 18 Desember 2015. [Internett].
Available: https://docs.opencv.org/3.1.0/dc/d2c/tutorial_real_time_pose.html.
[Funnet Januar 2018].

[11] Wikipedia, «Levenberg–Marquardt algorithm,» 9 Januar 2018. [Internett].
Available: https://en.wikipedia.org/wiki/Perspective-n-Point.
[Funnet Januar 2018].

[12] OpenCV, «OpenCV,» 2018. [Internett].
Available: https://opencv.org/.
[Funnet Januar 2018].

[13] OpenCV, «License,» 2018. [Internett].
Available: https://opencv.org/license.html.
[Funnet Januar 2018].

[14] Stackoverflow, «Stackoverflow,» [Internett].
Available: https://stackoverflow.com/.
[Funnet Januar 2018].

[15] Github, GitHub inc, 2018. [Internett].
Available: https://github.com/.
[Funnet Januar 2018].

[16] D. H. Hanssen, Programmerbar Logisk Styring, Bergen: Fagbokforlaget, 2015.

[17] Beckhoff, «Beckhoff.com,» Beckhoff, Januar 2018. [Internett].
Available: http://www.beckhoff.com/default.asp?start/default.htm.
[Funnet Januar 2018].

[18] Store Norske leksikon, «TCP/IP,» Oktober 2017. [Internett].
Available: https://snl.no/TCP/IP.
[Funnet November 2017].

[19] Beckhoff, «ADS Communication,» [Internett].
Available:
https://infosys.beckhoff.com/english.php?content=../content/1033/bc9000/html/bt_ethernet%20ads%20potocols.htm&id=.
[Funnet Januar 2018].

[20] ABB, «ABB Robotics,» [Internett].
Available: http://new.abb.com/products/robotics.
[Funnet Januar 2018].

[21] ABB, «IRB 6600,» [Internett].
Available:
http://www02.abb.com/global/inabb/inabb509.nsf/0/579e92967cad8bd16525703b00306b3e/$file/Robotics+IRB6600.pdf.
[Funnet Januar 2018].

[22] L. Sciavicco og B. Siciliano, Modelling And Control of Robot Manipulators, Springer.

[23] ABB, «Defining the tool frame,» [Internett].
Available:
http://developercenter.robotstudio.com/BlobProxy/manuals/IRC5FlexPendantOpManual/doc99.html.
[Funnet 12 2017].

[24] G. Bergstrom, «Method for calibration of off-line generated robot program,»
Chakmers University of Technology, Göteborg, 2011.

[25] J. Hallenberg, «Robot Tool Center Point Calibration,»
Linkopings University, Linkoping, 2007.

[26] F. C. Parker og B. J. Martin,
«Robot Sensor Calibration: Solving AX=XB on the Euclidean Group,»
IEEE transactions on robotics and automation, vol. 10, nr. 5, pp. 717-721, 1994.

[27] R. Askeland, «Implementering av maskinsyn for montering av fasadeplater,»
23 Januar 2018. [Internett].
Available: https://www.youtube.com/watch?v=6JfOcpnPhUQ&t=5s.
[Funnet Januar 2018].

[28] G. A. Aspheim, K. Bjørlykstøl og Ø. Vatne,
«Utvikling av verktøy for handtering og montering av fasadeplater med robot,»
Universitetet I Agder, Grimstad, 2017.

[29] Logitech, «Logitech,» November 2017. [Internett].
Available: http://support.logitech.com/en_us/product/webcam-c260/specs.

[30] Basler, «Basler ace acA2500-14um,» Basler AG, 2017. [Internett].
Available: https://www.baslerweb.com/en/products/cameras/area-scan-cameras/ace/aca2500-14um/.
[Funnet Desember 2017].

[31] Edmund Optics, «Edmund Optics,» November 2017. [Internett].
Available: https://www.edmundoptics.com/imaging-lenses/fixed-focal-length-lenses/4mm-uc-series-fixed-focal-length-lens/.

[32] C. Dahms, «OpenCV 3 Windows 10 installation Tutorial,» 6 August 2017. [Internett].
Available:
https://github.com/MicrocontrollersAndMore/OpenCV_3_Windows_10_Installation_Tutorial.
[Funnet September 2017].

[33] Basler, «Getting Started with pylon and OpenCV,» November 2017. [Internett].
Available: https://www.baslerweb.com/fp-1476182890/media/downloads/documents/application_notes/AW00136101000_Getting_Started_with_pylon4_and_OpenCV.pdf.

[34] Steni, «Steni Color Technical Datasheet,» Januar 2018. [Internett].
Available:
http://www.steni.no/docs/steni.no/02_teknisk%20dokumentasjon/teknisk%20datablad/technical%20datasheet%20steni%20colour.pdf.
[Funnet Januar 2018].

[35] Lady Ada, «Adafruit VL53L0X Time of Flight Micro-LIDAR Distance Sensor Breakout,» Adafruit, 12 Juli 2017. [Internett].
Available: https://learn.adafruit.com/adafruit-vl53l0x-micro-lidar-distance-sensor-breakout/overview.
[Funnet Oktober 2018].

[36] S. Tørdal, «HandEyeParkMartin.m,» 9 Mars 2017. [Internett].
Available: https://github.com/sondre1988/matlab-functions/blob/master/src/HandEyeParkMartin.m.
[Funnet Januar 2018].

[37] O. Semeniuta, «Analysis of camera calibration with respect to measurement accuracy,» i *48th CIRP International Conference on Manufacturing Systems (CIRP CMS 2015)*, Ischia, 2015.

8 Vedlegg

A. Masteroppgave MAS500, høst 2017:

Tittel: Utvikling av Kontrollsystem og Instrumentering for Kamera Montert på Robot
Kontaktperson ved UiA: Geir Hovland
Kontaktperson ved MacGregor Norway AS: Arne Tomstad

Ved Universitetet i Agder ble det i løpet av våren 2017 utviklet et verktøy i et Bachelorprosjekt for en industrirobot for å håndtere plater som skal monteres på en vegg, se bilder nedenfor.

Figur 52: Bilde av robotcelle etter alle platene er hengt opp.

Figur 53: Bilde av montering av plate på vegg.

I oppgaven ble et vanlig USB-kamera benyttet samt en LattePanda Single-Board-Computer (SBC) benyttet til bildebehandling. Denne SBC'en kjører både Windows med LabView for bildebehandling samt en Arduino for vakuumsensorer, lasermåler, og digitale signaler for overvåkning av verktøyets funksjoner.
Systemet som ble utviklet i det nevnte prosjektet fungerte rimelig bra, men det viste seg at kameraet hadde for dårlig oppløsning til at platene sin posisjon kunne estimeres nøyaktig nok.

I Masteroppgaven ønskes det at det jobbes med følgende:

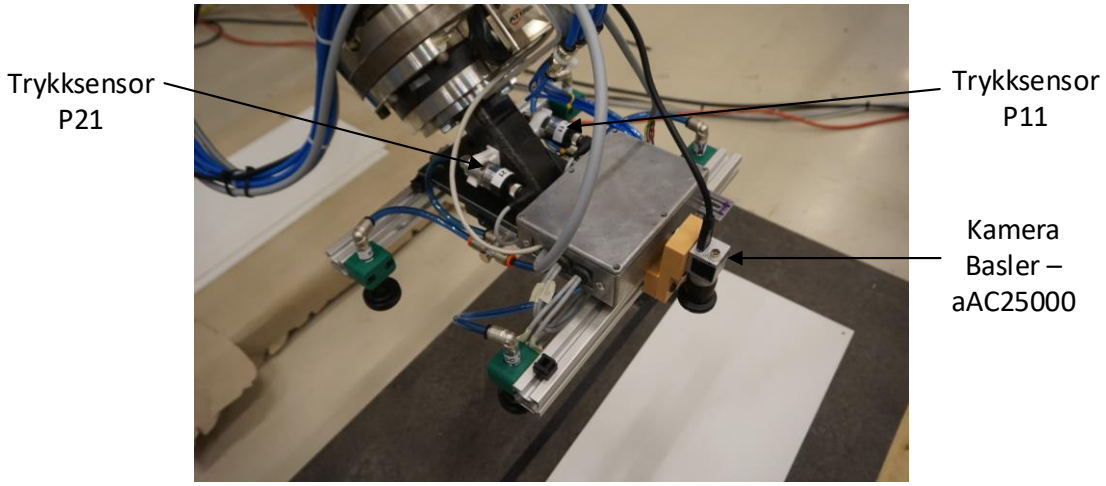
- Et eller flere 2D kamera (4K), vurdere IP kamera, bedre oppløsning enn det som ble brukt i Bachelorprosjektet våren 2017.
- Bruke en Beckhoff CX2040 som hovedkontroller til systemet og denne er plassert på bakken og ikke på verktøyhodet. Beckhoff PLC styrer all I/O og hovedfunksjoner slik at systemet blir mer ryddig.
- I stedet for spikere ønskes det at magneter (tape) benyttes for å feste platene på vegg
- Stendere vertikale eller horisontale) i stål (firkantrør, 50mm) benyttes. Senteravstand mellom stendere er 600mm.
- Lage et kalibreringspunkt på verktøyet.
- Lage en HMI som kjører på Beckhoff sin Embedded Windows PC.
- Det kan antas at den første platen allerede er montert på forhånd. De andre platene skal monteres relativt til denne.

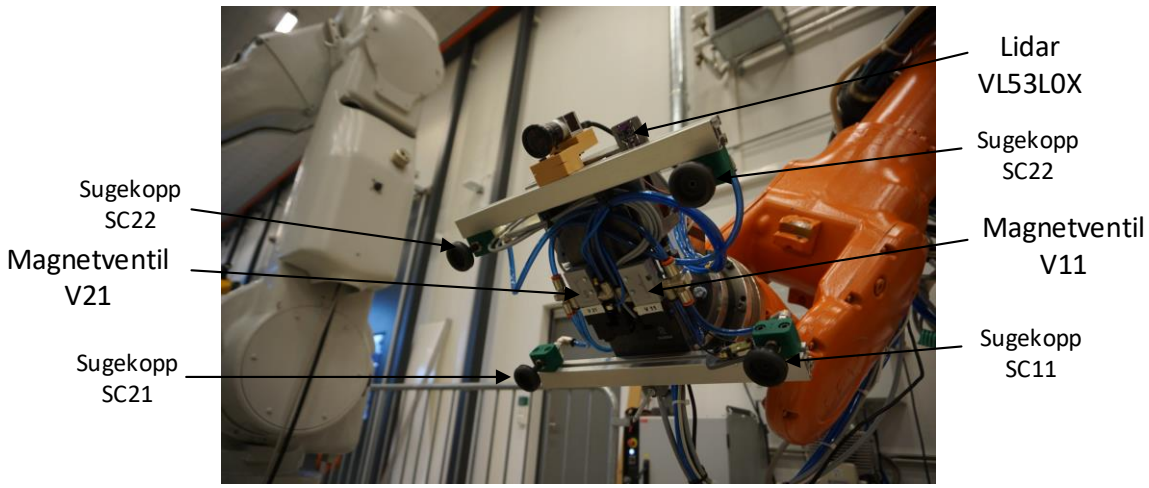
Viktige resultater å få svar på er:

- Få erfaring med bruk av 2D kamera og OpenCV implementert på Beckhoff.
- Bruk av OpenCV bibliotek med objekt gjenkjenning og kant detektering, og finne ut hvilken nøyaktighet dette gir.
- Bruke auto plassering av plater basert på deteksjon av minst en plate allerede plassert og definert avstand mellom platene.
- For å se realistiske resultater bør en vegg bygd med stendere brukes og stendere for å feste platene fast (i stedet for å henge opp platene på fastmonterte spikere slik som gjort i Bachelorprosjektet).
- Deteksjon av stendere inngår i oppgaven.
- Måle behandlingstid på Beckhoff kontroller ved de mest krevende bildebehandlings oppgavene for å få et inntrykk av om denne kontrolleren er kraftig nok for oppgaven.

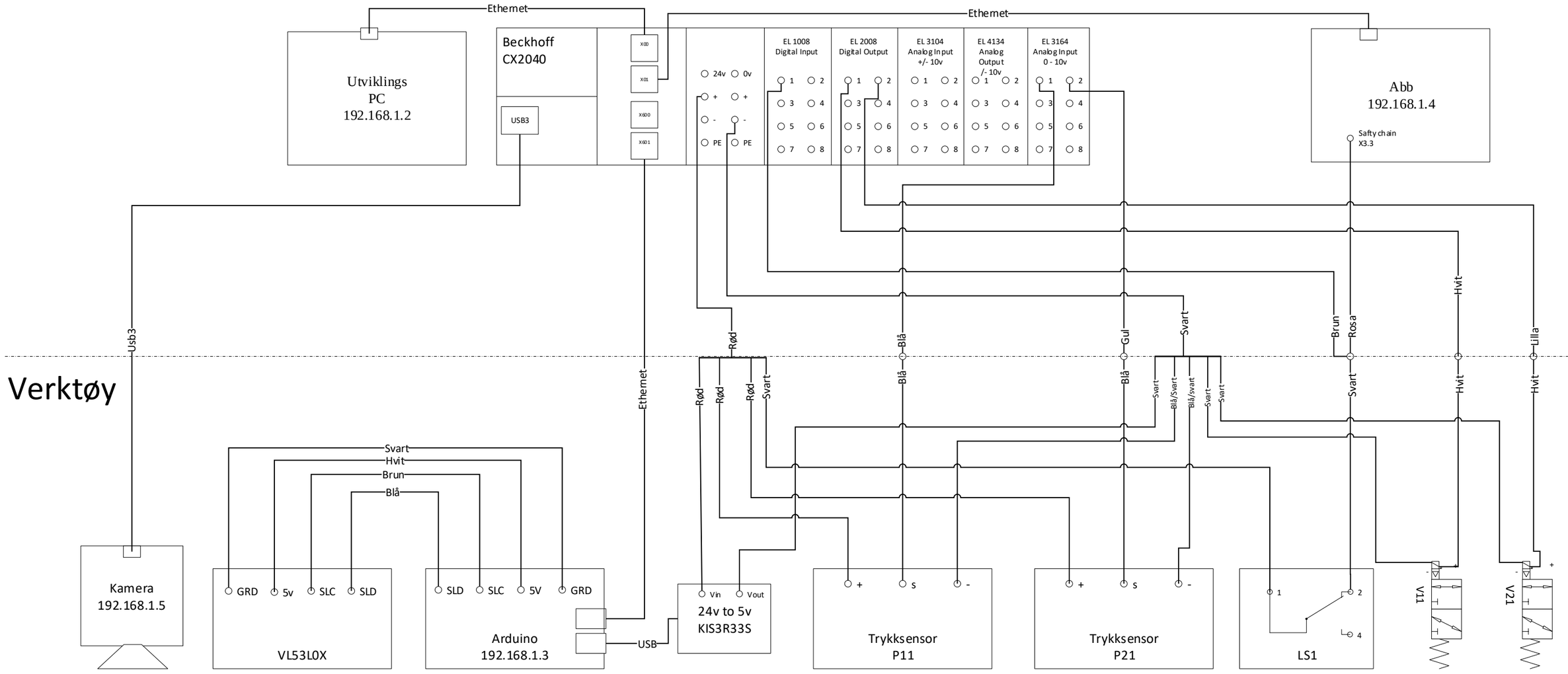
B. I/O liste

Komponent	System navn	Produsent	Model	Signal type	Inngang	Spenning	Tilkobling i CX2040
Trykksensor	P11	Festo	SPTW-B2R-G14-A-M-12	Analoge	Input	0-10V	EL-3164 Port: 1
Trykksensor	P21	Festo	SPTW-B2R-G14-A-M-12	Analoge	Input	0-10V	EL-3164 Port: 2
Magnetventil	V11	Festo	VUVS-L20-M32C-MZD-G18-F7V5	Digital	Output	0-24V	EL-2008 Port: 1
Magnetventil	V21	Festo	VUVS-L20-M32C-MZD-G18-F7V5	Digital	Output	0-24V	EL-2008 Port: 2
Trykk bryter	LS1			Digital	Input	0-24V	EL-1008 Port:1 IRC5 – X3.3





C. Koblingsskjema



D. Kode

Kode brukt i prosjektet kommer under i følgende rekkefølge:

PLS:

1. POU: Main
2. POU: TCP
3. POU: Vacuum
4. Global Variable List: GVL

Maskinsyn:

5. baslerToOpenCV.cpp

ABB Rapid:

6. platehaandler_h2017

Arduino:

7. Arduino

```
PROGRAM MAIN
VAR
    FBtcp           : TCP ;
    send            : E_send ;
    auto            : E_auto ;
    turn            : E_Turn ;
    sendmsg         : STRING ;
    FBvacuum        : vacuum ;
    coordsend       : STRING ;
    xs              : STRING        := 'x' ;
    ys              : STRING        := 'y' ;
    zs              : STRING        := 'z' ;
    axs             : STRING        := 'a' ;
    ays             : STRING        := 'b' ;
    azs             : STRING        := 'c' ;
    ts              : STRING        := 't' ;
    ns              : STRING        := 'n' ;
    recCoord        : STRING ;
    X               : REAL ;
    Y               : REAL ;
    Z               : REAL ;
    AngelX          : REAL ;
    plateTurn       : INT           := 0 ;
    plateNumber     : INT           := 1 ;
    vision          : BOOL ;
    platsOnPallet   : INT ;
    recCoordhold    : STRING ;
    xpos            : INT ;
    ypos            : INT ;
    zpos            : INT ;
    axpos           : INT ;
    aypos           : INT ;
    azpos           : INT ;
    xrob            : REAL ;
    yrob            : REAL ;
    zrob            : REAL ;
    axrob           : REAL ;
    ayrob           : REAL ;
    azrob           : REAL ;
    xrobs           : STRING ;
    lenofstring     : INT ;
    wallpos         : BOOL ;
    AngelY          : REAL ;
    AngelZ          : REAL ;
    counter         : INT ;
END_VAR
```

```
FBtcp ( sSend := sendmsg , sReceive => recCoord ) ;
FBvacuum ( ) ;

// -----------Create sting to send to IRC5------------------
coordsend := '' ;
coordsend := CONCAT ( coordsend , xs ) ;
coordsend := CONCAT ( coordsend , REAL_TO_STRING ( X ) ) ;
coordsend := CONCAT ( coordsend , ys ) ;
```

```
 9      coordsend  :=  CONCAT ( coordsend , REAL_TO_STRING ( Y ) ) ;
10      coordsend  :=  CONCAT ( coordsend , zs ) ;
11      coordsend  :=  CONCAT ( coordsend , REAL_TO_STRING ( Z ) ) ;
12      coordsend  :=  CONCAT ( coordsend , axs ) ;
13      coordsend  :=  CONCAT ( coordsend , REAL_TO_STRING ( AngelX ) ) ;
14      coordsend  :=  CONCAT ( coordsend , ays ) ;
15      coordsend  :=  CONCAT ( coordsend , REAL_TO_STRING ( AngelY ) ) ;
16      coordsend  :=  CONCAT ( coordsend , azs ) ;
17      coordsend  :=  CONCAT ( coordsend , REAL_TO_STRING ( AngelZ ) ) ;
18      coordsend  :=  CONCAT ( coordsend , ts ) ;
19      coordsend  :=  CONCAT ( coordsend , INT_TO_STRING  ( plateTurn ) ) ;
20      coordsend  :=  CONCAT ( coordsend , ns ) ;
21      coordsend  :=  CONCAT ( coordsend , INT_TO_STRING  ( plateNumber ) ) ;
22
23      //------- Move to Home position -----------------
24      IF  GVL . home  =  TRUE  THEN
25              CASE  send  OF
26                  E_Send . send :
27                  IF  GVL . home  =  TRUE  THEN
28                      sendmsg  :=  'Home' ;
29                      GVL . tcpcom  :=  TRUE ;
30                      send :=  E_Send . wait ;
31                  END_IF
32
33                  E_Send . wait :
34                  IF  GVL . tcpcom  =  FALSE  THEN
35                      sendmsg  :=  '' ;
36                      GVL . home  :=  FALSE ;
37                      send :=  E_Send . send ;
38                  END_IF
39
40              END_CASE
41          END_IF
42
43      // ------ Move to big pallet ----------
44          IF  GVL . bigp  =  TRUE  THEN
45              CASE  send  OF
46                  E_Send . send :
47                  IF  GVL . bigp  =  TRUE  THEN
48                      sendmsg  :=  'Go to big pallet' ;
49                      Wallpos  :=  FALSE ;
50                      GVL . tcpcom  :=  TRUE ;
51                      send :=  E_Send . wait ;
52                  END_IF
53
54                  E_Send . wait :
55                  IF  GVL . tcpcom  =  FALSE  THEN
56                      sendmsg  :=  '' ;
57                      GVL . bigp  :=  FALSE ;
58                      send :=  E_Send . send ;
59                  END_IF
60
61              END_CASE
62          END_IF
63
64      // --------Pick Plate ---------------
```

```
65      IF GVL.pickPlate = TRUE THEN
66              CASE send OF
67                  E_Send.send:
68                  IF GVL.pickPlate = TRUE THEN
69                      sendmsg := 'Pick plate';
70                      GVL.tcpcom := TRUE;
71                      send := E_Send.wait;
72                  END_IF
73
74                  E_Send.wait:
75                  IF GVL.tcpcom = FALSE THEN
76                      sendmsg := '';
77                      IF wallpos = TRUE THEN
78                      GVL.vacuum := TRUE ;
79                      GVL.pickPlate := FALSE;
80                      send := E_Send.send;
81                      ELSE
82                      GVL.vacuum := TRUE ;
83                      plateNumber := plateNumber + 1 ;
84                      GVL.pickPlate := FALSE;
85                      send := E_Send.send;
86                      END_IF
87
88                  END_IF
89
90              END_CASE
91          END_IF
92
93
94      // ------ Move to small pallet ----------
95          IF GVL.smallp = TRUE THEN
96              CASE send OF
97                  E_send.send:
98                  IF GVL.smallp = TRUE THEN
99                      sendmsg := 'Go to small pallet';
100                     GVL.tcpcom := TRUE;
101                     send := E_send.wait;
102                 END_IF
103
104                 E_send.wait:
105                 IF GVL.tcpcom = FALSE THEN
106                     sendmsg := '';
107                     GVL.vacuum := FALSE;
108                     GVL.smallp := FALSE;
109                     send := E_send.send;
110                 END_IF
111
112             END_CASE
113         END_IF
114
115     // ------ Move to wall1 ----------
116         IF GVL.wall = TRUE THEN
117             CASE send OF
118                 E_send.send:
119                 IF GVL.wall = TRUE THEN
120                     sendmsg := 'Wall';
```

```
121                     GVL . tcpcom  :=  TRUE ;
122                     send  :=  E_send . wait ;
123                 END_IF
124
125                 E_send . wait :
126                 IF  GVL . tcpcom  =  FALSE  THEN
127                     sendmsg  :=  '' ;
128                     GVL . Wall  :=  FALSE ;
129                     Wallpos  :=  TRUE ;
130                     send :=  E_send . send ;
131                 END_IF
132
133             END_CASE
134         END_IF
135
136
137         // ------ Move away from wall ----------
138         IF  GVL . wallaway  =  TRUE  THEN
139             CASE  send  OF
140                 E_send . send :
141                 IF  GVL . wallaway  =  TRUE  THEN
142                     sendmsg  :=  'Wall away' ;
143                     GVL . tcpcom  :=  TRUE ;
144                     send  :=  E_send . wait ;
145                 END_IF
146
147                 E_send . wait :
148                 IF  GVL . tcpcom  =  FALSE  THEN
149                     sendmsg  :=  '' ;
150                     GVL . Wallaway  :=  FALSE ;
151                     send :=  E_send . send ;
152                 END_IF
153
154             END_CASE
155         END_IF
156
157
158     // -------- Send Cordinats ----------------------------
159     IF  GVL . coordinates  =  TRUE  AND  Vision  =  FALSE  THEN
160             CASE  send  OF
161                 E_Send . send :
162                 IF  GVL . coordinates  =  TRUE  THEN
163                     sendmsg  :=  'Coordinates' ;
164                     GVL . tcpcom  :=  TRUE ;
165                     send :=  E_Send . wait ;
166                 END_IF
167
168                 E_Send . wait :
169                 IF  GVL . tcpcom  =  FALSE  THEN
170                     sendmsg  :=  '' ;
171                     send :=  E_Send . send2 ;
172                 END_IF
173
174                 E_Send . send2 :
175                 sendmsg  :=  coordsend ;
176                 GVL . tcpcom  :=  TRUE ;
```

```
177                 send := E_Send.wait2;
178
179                 E_Send.wait2:
180                 IF GVL.tcpcom = FALSE THEN
181                     sendmsg := '';
182                     GVL.coordinates := FALSE;
183                     send := E_Send.send;
184                 END_IF
185             END_CASE
186         END_IF
187
188
189
190     // ------- Recive coordinats -----------
191         IF GVL.recCoord = TRUE THEN
192             CASE send OF
193                 E_send.send:
194                 IF GVL.recCoord = TRUE THEN
195                     sendmsg := 'Robot position';
196                     GVL.tcpcom := TRUE;
197                     send := E_send.wait;
198                 END_IF
199
200                 E_send.wait:
201                 IF FIND(recCoord,'x') = 1 THEN
202                     recCoordhold := recCoord;
203                     xpos := FIND(recCoordhold,'x');
204                     ypos := FIND(recCoordhold,'y');
205                     zpos := FIND(recCoordhold,'z');
206                     axpos := FIND(recCoordhold,'ax');
207                     aypos := FIND(recCoordhold,'ay');
208                     azpos := FIND(recCoordhold,'az');
209
210                     xrob := STRING_TO_REAL(MID(recCoordhold, ypos-xpos-1,
    xpos+1));
211                     yrob := STRING_TO_REAL(MID(recCoordhold, zpos-ypos-1,
    ypos+1));
212                     zrob := STRING_TO_REAL(MID(recCoordhold, axpos-zpos-1,
    zpos+1));
213                     axrob := STRING_TO_REAL(MID(recCoordhold, aypos-axpos-1,
     axpos+2));
214                     ayrob := STRING_TO_REAL(MID(recCoordhold, azpos-aypos-1,
     aypos+2));
215                     lenofstring := LEN(recCoordhold);
216                     azrob := STRING_TO_REAL(MID(recCoordhold, lenofstring-
    azpos-1, azpos+2));
217                     GVL.exeption := TRUE;
218                     SEND := E_send.send2;
219                 END_IF
220
221                 E_send.send2:
222                 IF GVL.tcpcom = FALSE THEN
223                     sendmsg := '';
224                     GVL.exeption := FALSE;
225                     GVL.recCoord := FALSE;
226                     vision := TRUE;
```

```
227                     GVL . coordinates  :=  TRUE ;
228                     send :=  E_send . send ;
229                 END_IF
230
231             END_CASE
232         END_IF
233
234
235
236     // -------- Stop opperation -------------------------
237     IF  GVL . stop  =  TRUE  THEN
238             CASE  send  OF
239                 E_send . send :
240                 IF  GVL . stop  =  TRUE  THEN
241                     sendmsg  :=  'Stop' ;
242                     GVL . tcpcom  :=  TRUE ;
243                     send  :=  E_send . wait ;
244                 END_IF
245
246                 E_send . wait :
247                 IF  GVL . tcpcom  =  FALSE  THEN
248                     sendmsg  :=  '' ;
249                     GVL . stop  :=  FALSE ;
250                     GVL . disconnect  :=  TRUE ;
251                     send :=  E_send . send ;
252                 END_IF
253
254             END_CASE
255     END_IF
256
257     //-----------AUTO FUNCTION PLATE STORAGE TO TESTPALET
        ----------------------------
258         IF  GVL . auto  =  TRUE  THEN
259             CASE  auto  OF
260
261                 E_Auto . auto1 :
262                 IF  GVL . auto  =  TRUE  THEN
263                     GVL . vacuum  :=  FALSE ;
264                     GVL . bigp  :=  TRUE ;
265                     auto  :=  E_auto . auto2 ;
266                 END_IF
267
268                 E_Auto . auto2 :
269                 IF  GVL . bigp  =  FALSE  THEN
270                     GVL . recCoord  :=  TRUE  ;
271                     auto  :=  E_auto . auto3 ;
272                 END_IF
273
274                 E_Auto . auto3 :
275                 IF  GVL . recCoord  =  FALSE  THEN
276                     GVL . coordinates  :=  TRUE  ;
277                     auto  :=  E_auto . auto4 ;
278                 END_IF
279
280                 E_Auto . auto4 :
281                 IF  GVL . coordinates  =  FALSE  THEN
```

```
282                     GVL . pickPlate  :=  TRUE  ;
283                     auto  :=  E_auto . auto5 ;
284                 END_IF
285
286                 E_Auto . auto5 :
287                 IF  GVL . pickPlate  =  FALSE  THEN
288                     GVL . smallp  :=  TRUE  ;
289                     auto  :=  E_auto . auto6 ;
290                 END_IF
291
292                 E_Auto . auto6 :
293                 IF  GVL . smallp  =  FALSE  THEN
294
295                     IF  plateNumber  >  platsOnPallet  THEN
296                     GVL . auto  :=  FALSE ;
297                     GVL . home  :=  TRUE ;
298                     auto  :=  E_auto . auto1 ;
299                     ELSE  auto  :=  E_auto . auto1 ;
300                     END_IF
301                 END_IF
302
303             END_CASE
304         END_IF
305         //-----------AUTO FUNCTION PLATE STORAGE TO
        WALL----------------------------
306         IF  GVL . auto2  =  TRUE  THEN
307             CASE  auto  OF
308
309                 E_Auto . auto1 :
310                 IF  GVL . auto2  =  TRUE  THEN
311                     GVL . vacuum  :=  FALSE ;
312                     GVL . bigp  :=  TRUE ;
313                     auto  :=  E_auto . auto2 ;
314                 END_IF
315
316                 E_Auto . auto2 :
317                 IF  GVL . bigp  =  FALSE  THEN
318                     GVL . recCoord  :=  TRUE  ;
319                     auto  :=  E_auto . auto3 ;
320                 END_IF
321
322                 E_Auto . auto3 :
323                 IF  GVL . recCoord  =  FALSE  THEN
324                     GVL . coordinates  :=  TRUE  ;
325                     auto  :=  E_auto . auto4 ;
326                 END_IF
327
328                 E_Auto . auto4 :
329                 IF  GVL . coordinates  =  FALSE  THEN
330                     GVL . pickPlate  :=  TRUE  ;
331                     auto  :=  E_auto . auto5 ;
332                 END_IF
333
334                 E_Auto . auto5 :
335                 IF  GVL . pickPlate  =  FALSE  THEN
336                     GVL . wall  :=  TRUE  ;
```

```
337                     auto  :=  E_auto . auto6 ;
338                 END_IF
339
340                 E_Auto . auto6 :
341                 IF  GVL . wall  =  FALSE  THEN
342                     GVL . recCoord  :=  TRUE  ;
343                     auto  :=  E_auto . auto7 ;
344                 END_IF
345
346                 E_Auto . auto7 :
347                 IF  GVL . recCoord  =  FALSE  THEN
348                     GVL . coordinates  :=  TRUE  ;
349                     auto  :=  E_auto . auto8 ;
350                 END_IF
351
352                 E_Auto . auto8 :
353                 IF  GVL . coordinates  =  FALSE  THEN
354                     GVL . pickPlate  :=  TRUE  ;
355                     auto  :=  E_auto . auto9 ;
356                 END_IF
357
358                 E_Auto . auto9 :
359                 IF  GVL . pickplate  =  FALSE  THEN
360                     GVL . vacuum  :=  FALSE ;
361                     GVL . wallaway  :=  TRUE  ;
362                     auto  :=  E_auto . auto10 ;
363                 END_IF
364
365                 E_Auto . auto10 :
366                 IF  GVL . wallaway  =  FALSE  THEN
367                     GVL . home  :=  TRUE  ;
368                     GVL . auto2  :=  False ;
369                     auto  :=  E_auto . auto1 ;
370                 END_IF
371
372
373             END_CASE
374         END_IF
375
```

```
1   FUNCTION_BLOCK TCP
2   VAR_INPUT
3       sSend               : STRING ;  (* String to be sent *)
4   END_VAR
5   VAR_OUTPUT
6       sReceive            : STRING ;  (* String received *)
7   END_VAR
8   VAR
9       NetId               : STRING := '' ;   (* Server AmsNetId *)
10      ServerAdr           : STRING := '192.168.1.4' ;    (* Server IP address
    *)
11      ServerPort          : UDINT := 5000 ;  (* Server port number *)
12      fbSocketSend        : FB_SocketSend ;
13      fbSocketReceive     : FB_SocketReceive ;
14      bInitCloseAll       : BOOL := TRUE ;
15
16
17      bBusy               : BOOL ;
18      err                 : BOOL ;
19      errid               : UDINT ;
20      Timeout             : TIME := T#20S ;
21      fbSocketConnect     : FB_SocketConnect ;
22      hListen             : T_HSOCKET ;
23      hSocket             : T_HSOCKET ;
24      bConnAccepted       : BOOL ;
25      RxBuffer            : STRING ;
26      state               : E_State ;
27      com                 : E_com ;
28      disconnect          : E_Disconnect ;
29      fbSocketCloseAll    : FB_SocketCloseAll ;
30  END_VAR
31
```

```
1   fbSocketConnect (
2       sSrvNetId := NetId ,
3       sRemoteHost := ServerAdr ,
4       nRemotePort := ServerPort ,
5       bExecute := ,
6       tTimeout := T#5S ,
7       bBusy => bBusy ,
8       bError => err ,
9       nErrId => errid ,
10      hSocket => hSocket ) ;
11
12      fbSocketSend (
13          sSrvNetId := NetId ,
14          hSocket := hSocket ,
15          cbLen := INT_TO_UDINT ( LEN ( sSend ) ) , //LEN(sSend),
16          pSrc := ADR ( sSend ) , //ADR(sSend),
17          bExecute := ,
18          tTimeout := Timeout ,
19          bBusy => bBusy ,
20          bError => err ,
21          nErrId => errid ) ;
22
23  fbSocketReceive (
```

```
24              sSrvNetId := NetId,
25              hSocket := hSocket,
26              cbLen := SIZEOF(RxBuffer),
27              pDest := ADR(RxBuffer),
28              bExecute := ,
29              tTimeout := Timeout,
30              bBusy => bBusy,
31              bError => err,
32              nErrId => errid,
33              nRecBytes => );
34
35      fbSocketCloseAll(
36          sSrvNetId := NetId ,
37          bExecute := ,
38          tTimeout := Timeout,
39          bBusy => ,
40          bError => ,
41          nErrId => );
42
43
44
45
46      // -------------connection to IRC5----------------------
47      IF GVL.connect = TRUE THEN
48          CASE state OF
49
50              E_State.STATE_INIT:
51              IF GVL.connect = TRUE THEN
52              //Open the port
53              fbSocketConnect.bExecute := TRUE;
54              state := E_State.STATE_CONNECTING;
55              END_IF
56
57          E_State.STATE_CONNECTING:
58              fbSocketConnect.bExecute := FALSE;
59              GVL.connect := FALSE ;
60              GVL.connectlight := TRUE;
61              state := E_State.STATE_INIT;
62          END_CASE
63      END_IF
64
65      IF fbSocketConnect.bError = TRUE THEN;
66          GVL.connectlight := FALSE;
67      END_IF
68
69
70      //------ Send one message and recive 2 in return -------------
71      IF GVL.tcpcom = TRUE THEN
72              CASE com OF
73                  E_com.send:
74                  IF GVL.tcpcom = TRUE THEN
75                  fbSocketSend.bExecute := TRUE;
76                  com := E_com.recive;
77                  END_IF
78
79                  E_com.recive:
```

```
80                  fbSocketSend . bExecute  :=  FALSE ;
81                  IF  NOT  fbSocketSend . bBusy  AND  NOT  fbSocketSend . bError  THEN
82                  fbSocketReceive . bExecute :=  TRUE ;
83                  com  :=  E_com . wait ;
84                  END_IF
85
86
87                  E_com . wait :
88                  fbSocketReceive . bExecute  :=  FALSE ;
89                  IF  NOT  fbSocketReceive . bBusy  AND  NOT  fbSocketReceive . bError
        THEN
90                      IF  fbSocketReceive . nRecBytes  =  0  THEN
91                      //if there's no data received, reset the fbSocketReceive
92                      fbSocketReceive . bExecute  :=  TRUE ;
93                  ELSE
94                      MEMSET ( ADR ( sReceive ) , 0 , SIZEOF ( sReceive ) ) ;
95                      MEMCPY ( ADR ( sReceive ) ,  fbSocketReceive . pDest ,
        fbSocketReceive . nRecBytes ) ;
96                      com  :=  E_com . recive2 ;
97                  END_IF
98                  END_IF
99
100                 E_com . recive2 :
101                 IF  sSend  =  sReceive  THEN
102                     sReceive  := '' ;
103                     fbSocketReceive . bExecute  :=  TRUE ;
104                     com  :=  E_com . wait2 ;
105                 END_IF
106
107                 E_com . wait2 :
108                 fbSocketReceive . bExecute  :=  FALSE ;
109                 IF  NOT  fbSocketReceive . bBusy  AND  NOT  fbSocketReceive . bError
        THEN
110                     IF  fbSocketReceive . nRecBytes  =  0  THEN
111                     //if there's no data received, reset the fbSocketReceive
112                     fbSocketReceive . bExecute  :=  TRUE ;
113                 ELSE
114                     MEMSET ( ADR ( sReceive ) , 0 , SIZEOF ( sReceive ) ) ;
115                     MEMCPY ( ADR ( sReceive ) ,  fbSocketReceive . pDest ,
        fbSocketReceive . nRecBytes ) ;
116                     com  :=  E_com . recive3 ;
117                 END_IF
118                 END_IF
119
120                 E_com . recive3 :
121                 IF  sReceive  =  sSend  THEN
122                     sSend  :=  '' ;
123                     sReceive  := '' ;
124                     GVL . tcpcom  :=  FALSE ;
125                     com := E_com . send ;
126                 ELSIF  GVL . exeption  =  TRUE  THEN
127                     sSend  :=  '' ;
128                     sReceive  := '' ;
129                     GVL . tcpcom  :=  FALSE ;
130                     com := E_com . send ;
131                 END_IF
```

```
132
133             END_CASE
134             END_IF
135
136     // ---------------- Close all sockets ----------------------------------
137     IF GVL.disconnect = TRUE THEN
138
139     CASE disconnect OF
140         E_Disconnect.disconnect:
141         IF GVL.disconnect = TRUE THEN
142             fbsocketCloseAll.bExecute := TRUE;
143             disconnect := E_Disconnect.wait;
144         END_IF
145
146         E_Disconnect.wait:
147         fbsocketCloseAll.bExecute := FALSE;
148         IF NOT fbsocketCloseAll.bBusy OR NOT fbsocketCloseAll.bError THEN
149             GVL.disconnect := FALSE;
150             GVL.connectlight := FALSE;
151             disconnect := E_Disconnect.disconnect;
152         END_IF
153
154     END_CASE
155     END_IF
156
157
158
159
```

```
1   FUNCTION_BLOCK              vacuum
2
3   VAR_INPUT
4   END_VAR
5
6   VAR_OUTPUT
7       suctionStatus           : BOOL ;
8   END_VAR
9   VAR
10      p11Volt                 : REAL ;
11      p21Volt                 : REAL  ;
12      voltLimit               : REAL := 2 ;
13      led1                    : BOOL ;
14      led2                    : BOOL ;
15
16
17      v11Write AT %Q*             : BOOL ; //v21Write : vacuum valve v11 True
    når den ikke er åpen
18      v21Write AT %Q*             : BOOL ; //v21Write : vacuum valve v21,
    True når den ikke er åpen
19      ls1Read AT %I*          : BOOL ; //ls1Read : Tool contact sensor LS1,
20      p11Read AT %I*          : INT ; //p11Read : vacuum sensor p11
21      p21Read AT %I*          : INT ; //p11read : vacuum sensor p21
22  END_VAR
23
```

```
1   p11Volt := p11Read * 10 ;
2   p11Volt := p11Volt / 32767 ;
3   IF p11Volt < voltLimit THEN
4       led1 := TRUE ;
5       ELSE led1 := FALSE ;
6   END_IF
7
8
9   p21Volt := p21Read * 10 ;
10  p21Volt := p21Volt / 32767 ;
11  IF p21Volt < voltLimit THEN
12      led2 := TRUE ;
13      ELSE led2 := FALSE ;
14  END_IF
15
16
17  IF GVL.vacuum = TRUE
18      THEN v11Write := FALSE ; v21Write := FALSE ;
19      ELSe v11Write := TRUE ; v21Write := TRUE ;
20  END_IF
21
22  IF GVL.vacuum = TRUE AND p11Volt < voltLimit AND p21Volt < voltLimit
23      THEN GVL.eStop := FALSE ;
24          ELSIF GVL.vacuum = TRUE AND p11Volt >= voltLimit AND p21Volt <
     voltLimit
25      THEN  GVL.eStop := TRUE ;
26          ELSIF GVL.vacuum = TRUE AND p11Volt < voltLimit AND p21Volt >=
     voltLimit
27      THEN GVL.eStop := TRUE ;
28          ELSIF GVL.vacuum = TRUE AND p11Volt >= voltLimit AND p21Volt
```

```
        >= voltLimit
29          THEN   GVL . eStop  := TRUE  ;
30      ELSE   GVL . eStop  := FALSE ;
31      END_IF
32
33      GVL . AlarmLED  :=  ls1Read ;
34
```

```
1   {attribute 'qualified_only'}
2   VAR_GLOBAL
3       connect             : BOOL ;
4       tcpcom              : BOOL ;
5       connectlight        : BOOL ;
6       auto                : BOOL ;
7       auto2               : BOOL ;
8       bigp                : BOOL ;
9       smallp              : BOOL ;
10      vacuum              : BOOL                  := FALSE ;
11      eStop               : BOOL ;
12      turn                : BOOL ;
13      disconnect          : BOOL ;
14      stop                : BOOL ;
15      coordinates         : BOOL ;
16      pickPlate           : BOOL ;
17      home                : BOOL ;
18      exeption            : BOOL ;
19      wall                : BOOL ;
20      wallaway            : BOOL ;
21      recCoord            : BOOL ;
22      AlarmLED            : BOOL ;
23  END_VAR
24
```

```
/*

Masteroppgave: Implementering av maskinsyn for montering av fasadeplater
Fag: MAS 500
Forfatter: Remi Askeland

Universitetet i Agder, 2018
Fakultet for teknologi og realfag
Institutt for ingeniørvitenskap


Dette scripte tar et bilde med et Basler aAc2500 kamera,
konverterer bilde til OpenCV format.
Bilde blir så prosesert for og finne senter og rotasjon av et rektangel.
Posisjonen og vinkelen blir sendt videre til en Beckhoff PLC

*/


#define saveImages 0  // Define if images are to be saved. '0'- no; '1'- yes.

#define imgShow 1           // 1 show original vs undistored image vs threshold image

#define visualThreshold 0   // 1 show the diferent thresholding opptions

#define solvePnPMethode 1   //  not in use

#define imageHold 0         // 1 to not remove image after use

// Debuging
#define PR_debug  0         // 1 - Show all log messages
#define PRtime_debug  0     // 1 - Show all logg time

#if PR_debug == 1
#define LOG(x,y) std::cout << x << std::endl << y << std::endl
#else
#define LOG(x,y)
#endif

#if PRtime_debug == 1
#define LOGtime(name) std::cout << name <<  t << " seconds." << std::endl;
#else
#define LOGtime(name)
#endif

// ------------------ Libarys used -------------------------
#include<stdio.h>
#define _WINSOCK_DEPRECATED_NO_WARNINGS
#include<winsock2.h>
#pragma comment(lib,"ws2_32.lib") //Winsock Library

#include <stdio.h>
#include <stdlib.h>
#include "windows.h"
#include <iostream>
```

```
 56 using namespace std;
 57
 58 // Include files to use OpenCV API.
 59 #include <opencv2/core/core.hpp>
 60 #include <opencv2/highgui/highgui.hpp>
 61 #include <opencv2/video/video.hpp>
 62 #include "opencv2/imgproc/imgproc.hpp"
 63 #include "opencv2/features2d/features2d.hpp"
 64 #include "opencv2/calib3d/calib3d.hpp"
 65 #include <opencv2/core/eigen.hpp>
 66 using namespace cv;
 67 // Timing  var
 68 #define get_ticks
 69 #define get_freq
 70 // save to file lib
 71 #include <fstream>
 72 // Eigen lib
 73 #include <Eigen/Dense>
 74 #include <Eigen/Core>
 75 #include <Eigen/Geometry>
 76 using namespace Eigen;
 77
 78 // Include files to use the PYLON API.
 79 #include <pylon/PylonIncludes.h>
 80 #ifdef PYLON_WIN_BUILD
 81 #    include <pylon/PylonGUI.h>
 82 #endif
 83 using namespace Pylon;
 84 // ADS lib
 85 #include <TcAdsDef.h>
 86 #include <TcAdsAPI.h>
 87
 88 // ----------------------Calibration data and camera                    ↵
      intrinsics----------------------
 89 // Calibration 11.12 big plate
 90 double fx = 1.8537e3; // focal lengt expressed in pixels
 91 double fy = 1.8562e3;
 92 double cx = 1299.5; // Principal point
 93 double cy = 1025.5;
 94 double skew = 0;
 95 double k1 = -0.2608; //  radial factor
 96 double k2 = 0.1386;
 97 double k3 = -0.0425;
 98 double p1 = 3.4612e-5; // tangential factor
 99 double p2 = 1.0446e-4;
100 double my = 2.2e-6; //pixel size
101 double imageSizeY;
102 double imageSizeX;
103 //---------------Functions -------------
104 void showimg(string windowName, Mat imgVariabel);
105 void udp(void);
106 void Threshold_Demo(int, void*);
107 void extrafunctions(void);
108 void kinematics(void);
109 //--------------------Posistion estemation--------------------------------
110
```

```
111  static const uint32_t c_countOfImagesToGrab = 1;     // Number of images to be
        grabbed.
112  int imagecount = 0;                                  // Countas how may images
        is taken in this sesion
113  Mat openCvImage;                                     // Create an OpenCV
       image.
114
115  double u;
116  double v;
117  double x;
118  double y;
119  double z;
120  double zMeasured;
121  double t; // variable used in timing operations
122  double Xrob;
123  double Yrob;
124  double Zrob;
125  float angel;// Angel of rect
126
127  // Position of plate acording to base frame
128  double X; // X coordinate of detected object
129  double Y; // Y coordinate of detected object
130  double Z; // Z coordinate of detected object
131  double rotZ;
132  double rotY;
133  double rotX;
134
135
136
137  int thresholdValue;
138  int maxBINARYValue = 255;
139  int thresholdType = 0;
140  int  thresholdPlateStorage = 120;
141  int thresholdWall = 90;
142
143  Mat intrinsic = (Mat1d(3, 3) << fx, 0, cx, 0, fy, cy, 0, 0, 1);
144  Mat distortionCoefficients = (Mat1d(1, 5) << k1, k2, p1, p2, k3);
145  Mat img;
146  Mat imgUndistorted;
147  Mat imgGray;
148  Mat imgThreshold;
149  Mat drawing;
150
151  vector<vector<Point>> contours;
152  vector<Vec4i> hierarchy;
153  Point2f center;
154  cv::Scalar red(0, 0, 255);
155  cv::Scalar green(0, 255, 0);
156  cv::Scalar blue(255, 0, 0);
157  cv::Scalar black(0, 0, 0);
158  cv::Scalar white(255, 255, 255);
159  cv::Scalar gray(128, 128, 128);
160
161  // Variabels for threshold demo
162  int threshold_value = 0;
163  int threshold_type = 3;;
```

```
164 int const max_value = 255;
165 int const max_type = 4;
166 int const max_BINARY_value = 255;
167
168 double plateoffsetx;
169 double plateoffsety;
170 cv::Mat rvec(3, 1, cv::DataType<double>::type);
171 cv::Mat tvec(3, 1, cv::DataType<double>::type);
172 cv::Mat RotMat(3, 3, cv::DataType<double>::type);
173 Eigen::MatrixXd RotMatEigen;
174 double centeroffset;
175 cv::Mat rvecc(3, 1, cv::DataType<double>::type);
176 cv::Mat tvecc(3, 1, cv::DataType<double>::type);
177 //-------------------- ADS variables -----------------------------------
178 long      nErr, nPort;      //ADS port and error message
179 AmsAddr   Addr;             //ADS adress
180 PAmsAddr  pAddr = &Addr;
181
182 // Variabel to start vision script
183 long     lHdlVar; bool  nData;  char    szVar[] = { "MAIN.vision" };
184 // Variabel for X psition
185 char XVar[] = { "Main.X" };
186 long   lHdlVarX;
187 float   nDataX;
188 // Variabel for Y psition
189 char YVar[] = { "Main.Y" }; long   lHdlVarY; float  nDataY;
190 // Variabel for Z psition
191 char ZVar[] = { "Main.Z" }; long   lHdlVarZ; float  nDataZ;
192 // Variabel for Angel psition
193 char AngelXVar[] = { "Main.AngelX" }; long lHdlVarAngelX; float nDataAngelX;
194 char AngelYVar[] = { "Main.AngelY" }; long lHdlVarAngelY; float nDataAngelY;
195 char AngelZVar[] = { "Main.AngelZ" }; long lHdlVarAngelZ; float nDataAngelZ;
196 // Variabel for number off plates on pallet
197 char popVar[] = { "Main.platsOnPallet" };
198 long   lHdlVarpop;
199 int     nDatapop;
200 // Variabel for rob pos
201 char xrobVar[] = { "Main.xrob" }; long lHdlVarxrob; float xrob;
202 char yrobVar[] = { "Main.yrob" }; long lHdlVaryrob; float yrob;
203 char zrobVar[] = { "Main.zrob" }; long lHdlVarzrob; float zrob;
204 char axrobVar[] = { "Main.axrob" }; long   lHdlVaraxrob; float axrob;
205 char ayrobVar[] = { "Main.ayrob" }; long  lHdlVarayrob; float ayrob;
206 char azrobVar[] = { "Main.azrob" }; long   lHdlVarazrob; float azrob;
207 // Variabel for rob pos
208 char wallposVar[] = { "Main.wallpos" }; long lHdlVarwallpos; bool wallpos;
209
210 // ============================== MAIN                                    ↵
      ======================================================
211 int main()
212 {
213     cout << "Vision initializing start " << endl;
214     int exitCode = 0;       // The exit code of the application.
215     //kinematics();
216     // -------------------- ADS comunication ----------------------
217     std::ofstream foutsave;
218     foutsave.open("Master_Vision_Remi.txt");
```

```
219     // Open communication port on the ADS router
220     nPort = AdsPortOpen();
221     nErr = AdsGetLocalAddress(pAddr);
222
223     if (nErr) cerr << "Error: AdsGetLocalAddress: " << nErr << '\n';
224     pAddr->port = 851;
225
226     // Get the handle of the PLC-variable <szVar>
227     nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof
          (lHdlVar), &lHdlVar, sizeof(szVar), szVar);
228     nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof
          (lHdlVarX), &lHdlVarX, sizeof(XVar), XVar);
229     nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof
          (lHdlVarY), &lHdlVarY, sizeof(YVar), YVar);
230     nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof
          (lHdlVarZ), &lHdlVarZ, sizeof(ZVar), ZVar);
231     nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof
          (lHdlVarAngelX), &lHdlVarAngelX, sizeof(AngelXVar), AngelXVar);
232     nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof
          (lHdlVarAngelY), &lHdlVarAngelY, sizeof(AngelYVar), AngelYVar);
233     nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof
          (lHdlVarAngelZ), &lHdlVarAngelZ, sizeof(AngelZVar), AngelZVar);
234     nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof
          (lHdlVarpop), &lHdlVarpop, sizeof(popVar), popVar);
235     nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof
          (lHdlVarxrob), &lHdlVarxrob, sizeof(xrobVar), xrobVar);
236     nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof
          (lHdlVaryrob), &lHdlVaryrob, sizeof(yrobVar), yrobVar);
237     nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof
          (lHdlVaryrob), &lHdlVarzrob, sizeof(zrobVar), zrobVar);
238     nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof
          (lHdlVaraxrob), &lHdlVaraxrob, sizeof(axrobVar), axrobVar);
239     nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof
          (lHdlVarayrob), &lHdlVarayrob, sizeof(ayrobVar), ayrobVar);
240     nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof
          (lHdlVarazrob), &lHdlVarazrob, sizeof(azrobVar), azrobVar);
241     nErr = AdsSyncReadWriteReq(pAddr, ADSIGRP_SYM_HNDBYNAME, 0x0, sizeof
          (lHdlVarwallpos), &lHdlVarwallpos, sizeof(wallposVar), wallposVar);
242     if (nErr) cerr << "Error: AdsSyncReadWriteReq: " << nErr << '\n';
243
244     // ---------------------- Pylon image--------
245     // Automagically call PylonInitialize and PylonTerminate to ensure the
          pylon runtime system
246     // is initialized during the lifetime of this object.
247     Pylon::PylonAutoInitTerm autoInitTerm;
248
249     // Create an instant camera object with the camera device found first.
250     CInstantCamera camera(CTlFactory::GetInstance().CreateFirstDevice());
251
252     // Print the model name of the camera.
253     cout << "Using device " << camera.GetDeviceInfo().GetVendorName() << " "
          << camera.GetDeviceInfo().GetModelName() << endl;
254
255     // Get a camera nodemap in order to access camera parameters.
256     GenApi::INodeMap& nodemap = camera.GetNodeMap();
257     // Open the camera before accessing any parameters.
```

```
258     camera.Open();
259     // Create pointers to access the camera Width and Height parameters.
260     GenApi::CIntegerPtr width = nodemap.GetNode("Width");
261     GenApi::CIntegerPtr height = nodemap.GetNode("Height");
262
263     // The parameter MaxNumBuffer can be used to control the count of buffers
264     // allocated for grabbing. The default value of this parameter is 10.
265     camera.MaxNumBuffer = 5;
266
267     // Create a pylon ImageFormatConverter object.
268     CImageFormatConverter formatConverter;
269     // Specify the output pixel format.
270     formatConverter.OutputPixelFormat = PixelType_Mono8; //
        PixelType_Mono12;//PixelType_BGR8packed;  // Mono 8 format
271                                                          // Create a
                    PylonImage that will be used to create OpenCV images
                    later.
272     CPylonImage pylonImage;
273     // Declare an integer variable to count the number of grabbed images
274     // and create image file names with ascending number.
275     int grabbedImages = 0;
276
277
278     VideoWriter cvVideoCreator;
                // Create an OpenCV video creator
279     std::string videoFileName = "openCvVideo.avi";
                // Define the video file name
280     cv::Size frameSize = Size((int)width->GetValue(), (int)height->GetValue
        ());        // Define the video frame size
281
282
          // Set the codec type and the frame rate. You have 3 codec options
    here.
283
          // The frame rate should match or be lower than the camera
    acquisition frame rate.
284     cvVideoCreator.open(videoFileName, CV_FOURCC('D', 'I', 'V', 'X'), 14,
        frameSize, true);
285     //cvVideoCreator.open(videoFileName, CV_FOURCC('M','P','4','2'), 20,
        frameSize, true);
286     //cvVideoCreator.open(videoFileName, CV_FOURCC('M','J','P','G'), 20,
        frameSize, true);
287
288     cout << "Vision inializing complete" << endl << "" << endl;
289     while (1 == 1) {
290         // Read start variabel
291         nErr = AdsSyncReadReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVar, sizeof
            (nData), &nData);
292         nErr = AdsSyncReadReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVarpop, sizeof
            (nDatapop), &nDatapop);
293         nErr = AdsSyncReadReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVarxrob,
            sizeof(xrob), &xrob);
294         nErr = AdsSyncReadReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVaryrob,
            sizeof(yrob), &yrob);
295         nErr = AdsSyncReadReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVarzrob,
            sizeof(zrob), &zrob);
```

```
296         nErr = AdsSyncReadReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVaraxrob,
              sizeof(axrob), &axrob);
297         nErr = AdsSyncReadReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVarayrob,
              sizeof(ayrob), &ayrob);
298         nErr = AdsSyncReadReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVarazrob,
              sizeof(azrob), &azrob);
299         nErr = AdsSyncReadReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVarwallpos,
              sizeof(wallpos), &wallpos);
300         if (nErr) {
301             cerr << "Error: AdsSyncReadReq: " << nErr << '\n';
302             return exitCode;
303         }
304
305
306         if (nData == 1) {
307             //std::cout << "Prosseing" << std::endl;
308             std::cout<<"" <<std::endl;
309             destroyAllWindows();
310
311             double t = getTickCount(); //to time the undistort function
312             try
313             {
314                 // Start the grabbing of c_countOfImagesToGrab images.
315                 // The camera device is parameterized with a default
                      configuration which
316                 // sets up free-running continuous acquisition.
317                 camera.StartGrabbing(c_countOfImagesToGrab,
                      GrabStrategy_LatestImageOnly);
318
319                 // This smart pointer will receive the grab result data.
320                 CGrabResultPtr ptrGrabResult;
321
322                 // Camera.StopGrabbing() is called automatically by the
                      RetrieveResult() method
323                 // when c_countOfImagesToGrab images have been retrieved.
324                 while (camera.IsGrabbing())
325                 {
326                     // Wait for an image and then retrieve it. A timeout of
                        5000 ms is used.
327                     camera.RetrieveResult(5000, ptrGrabResult,
                        TimeoutHandling_ThrowException);
328
329                     // Image grabbed successfully?
330                     if (ptrGrabResult->GrabSucceeded())
331                     {
332                         // Access the image data.
333                         imageSizeX = ptrGrabResult->GetWidth();
334                         imageSizeY = ptrGrabResult->GetHeight();
335
336
337                         imagecount = imagecount + 1;
338                         cout << "Image counter:  " << imagecount << endl;
339
340
341                         LOG("Image sizeX: ", ptrGrabResult->GetWidth());
342                         LOG("Image sizeY: " , ptrGrabResult->GetHeight());
```

```
343
344
345                         // Convert the grabbed buffer to a pylon image.
346                         formatConverter.Convert(pylonImage, ptrGrabResult);
347
348                         // Create an OpenCV image from a pylon image.
349                         //openCvImage = cv::Mat(ptrGrabResult->GetHeight(),
                        ptrGrabResult->GetWidth(), CV_8UC1, (uint8_t *)
                        pylonImage.GetBuffer()); //CV_8UC1 - 8 bit unsigned
                        integer with 1 channel, use if mono 8
350                         // the above line creates the image container and
                        putts in the pixels in same operartions and the
                        camerapointer is locket
351                         openCvImage = cv::Mat(ptrGrabResult->GetHeight(),
                        ptrGrabResult->GetWidth(), CV_8UC1); //create the image
                        container
352                         memcpy(openCvImage.ptr(), pylonImage.GetBuffer(),
                        (ptrGrabResult->GetHeight())*(ptrGrabResult->GetWidth
                        ())); //putts the pixels in the image container
353
354
355                         // Set saveImages to '1' to save images.
356                         if (saveImages) {
357                             // Create the current image name for saving.
358                             std::ostringstream s;
359                             // Create image name files with ascending grabbed
                         image numbers.
360                             s << "image_" << grabbedImages << ".jpg";
361                             std::string imageName(s.str());
362                             // Save an OpenCV image.
363                             imwrite(imageName, openCvImage);
364                             grabbedImages++;
365                         }
366                     }
367                     else
368                     {
369                         cout << "Error: " << ptrGrabResult->GetErrorCode() <<
                         " " << ptrGrabResult->GetErrorDescription() << endl;
370                     }
371                 }
372
373             }
374             catch (GenICam::GenericException &e)
375             {
376                 // Error handling.
377                 cerr << "An exception occurred." << endl
378                     << e.GetDescription() << endl;
379                 exitCode = 1;
380             }
381
382             t = ((double)getTickCount() - t) / getTickFrequency();
383             LOGtime("Time for image collection = ");
384
385             // -------- Vision prosseing -----------------------------
386             img = openCvImage; //img used past this point
387             t = getTickCount(); //to time the undistort function
```

```
388             undistort(img, imgUndistorted, intrinsic,
                  distortionCoefficients); // undistorts the img
389             t = ((double)getTickCount() - t) / getTickFrequency();
390             LOGtime("Time for undistortion = ");
391
392             // -------------Threshold demo ------------------------
393
394             if (visualThreshold) {
395                 char* window_name = "Threshold Demo";
396                 char* trackbar_type = "Type: \n 0: Binary \n 1: Binary
                    Inverted \n 2: Truncate \n 3: To Zero \n 4: To Zero
                    Inverted";
397                 char* trackbar_value = "Value";
398                 /// Create a window to display results
399                 namedWindow(window_name, CV_WINDOW_NORMAL);
400                 resizeWindow(window_name, 800, 600);
401                 /// Create Trackbar to choose type of Threshold
402                 createTrackbar(trackbar_type, window_name, &threshold_type,
                    max_type, Threshold_Demo);
403                 createTrackbar(trackbar_value, window_name, &threshold_value,
                     max_value, Threshold_Demo);
404                 /// Call the function to initialize
405                 Threshold_Demo(0, 0);
406                 /// Wait until user finishes program
407                 while (true)
408                 {
409                     int c;
410                     c = waitKey(20);
411                     if ((char)c == 27)
412                     {
413                         break;
414                     }
415                 }
416             }
417
418             // ------------------- Image prossesing ------------------------
419             t = getTickCount(); //to time the image prossesing
420
421             //Cheks if robot is at wall or at pallet
422             // Different threshold walue becouse of different light
                  condintions
423             if (wallpos == 1) {
424                 thresholdValue = thresholdWall;
425                 centeroffset = 595;
426                 LOG("Wallpos = ", wallpos);
427                 std::cout << "Position: Wall" << std::endl;
428                 Mat imgblur;
429                 cv::GaussianBlur(imgUndistorted, imgblur, Size(15, 15), 0,
                    0);
430                 threshold(imgblur, imgThreshold, 0, 255, THRESH_BINARY |
                    THRESH_OTSU);
431                 //showimg("OTSU", imgThreshold);
432                 //showimg("blur", imgblur);
433
434             }
435             else {
```

```
436                 thresholdValue = thresholdPlateStorage;
437                 centeroffset = 0;
438                 LOG("Wallpos = ", wallpos);
439                 std::cout << "Position: Plate Storage" << std::endl;
440                 threshold(imgUndistorted, imgThreshold, thresholdValue,
                      maxBINARYValue, thresholdType);
441                 //showimg("Threshold", imgThreshold);
442             }
443
444             // find conturs colected from: https://stackoverflow.com/
                  questions/26583649/opencv-c-rectangle-detection-which-has-
                  irregular-side
445             // find contours(if always so easy to segment as your image, you
                  could just add the black / rect pixels to a vector)
446
447             findContours(imgThreshold, contours, CV_RETR_EXTERNAL,
                  CV_CHAIN_APPROX_SIMPLE);
448             vector<Vec4i> hierarchy;
449             Mat drawing = Mat::zeros(imgUndistorted.size(), CV_8UC3);
450             drawing.setTo(cv::Scalar(white));
451             /// Draw contours and find biggest contour (if there are other
                  contours in the image, we assume the biggest one is the desired
                   rect)
452             // drawing here is only for demonstration!
453             int biggestContourIdx = -1;
454             float biggestContourArea = 0;
455             for (int i = 0; i < contours.size(); i++)
456             {
457                 if (imgShow)
458                 {
459                     drawContours(drawing, contours, i, gray, 2, 8, hierarchy,
                          0, Point());
460                 }
461                 float ctArea = contourArea(contours[i]);
462                     if (ctArea > biggestContourArea)
463                     {
464                     biggestContourArea = ctArea;
465                     biggestContourIdx = i;
466                 }
467             }
468
469             // if no contour found
470             if (biggestContourIdx < 0)
471             {
472             cout << "no contour found" << std::endl;
473             return 1;
474             }
475             // compute the rotated bounding rect of the biggest contour!
                  (this is the part that does what you want/need)
476             RotatedRect boundingBox = minAreaRect(contours
                  [biggestContourIdx]);
477             // one thing to remark: this will compute the OUTER boundary box,
                   so maybe you have to erode/dilate if you want something
                  between the ragged lines
478             center = boundingBox.center;
479             // draw the rotated rect
```

```
480             Point2f corners[4];
481             boundingBox.points(corners);
482             std::vector<cv::Point2f> corners2;
483             corners2.push_back(corners[0]);
484             corners2.push_back(corners[1]);
485             corners2.push_back(corners[2]);
486             corners2.push_back(corners[3]);
487
488             // Angel of rect tangel, openCV only use -90 to 0
489             if (boundingBox.size.width < boundingBox.size.height) {
490                 angel = (90 + boundingBox.angle);
491                 LOG("Angle along longer side: " , angel);
492             }
493             else {
494                 angel = (boundingBox.angle);
495                 LOG("Angle along longer side: ", angel);
496             }
497
498             if (solvePnPMethode) {
499                 //// TEST av solve PNP
500
501                 double hight = 295;
502                 double width = 595;
503                 std::vector<cv::Point3f> objectPoints;
504
505                 if (boundingBox.size.height < boundingBox.size.width) {
506
507                     objectPoints.push_back(cv::Point3f(0, 0, 0));
508                     objectPoints.push_back(cv::Point3f(0, hight, 0));
509                     objectPoints.push_back(cv::Point3f(width, hight, 0));
510                     objectPoints.push_back(cv::Point3f(width, y, 0));
511                     plateoffsetx = width / 2;
512                     plateoffsety = hight / 2;
513                 }
514                 else {
515                     objectPoints.push_back(cv::Point3f(0, 0, 0));
516                     objectPoints.push_back(cv::Point3f(width, 0, 0));
517                     objectPoints.push_back(cv::Point3f(width, hight, 0));
518                     objectPoints.push_back(cv::Point3f(0, hight, 0));
519                     plateoffsetx = width / 2;
520                     plateoffsety = hight / 2;
521                 }
522
523                 cv::Mat cameraMatrix(3, 3, cv::DataType<double>::type);
524                 cv::setIdentity(cameraMatrix);
525                 cameraMatrix.at<double>(0, 0) = fx;
526                 cameraMatrix.at<double>(0, 2) = cx;
527                 cameraMatrix.at<double>(1, 1) = fy;
528                 cameraMatrix.at<double>(1, 2) = cy;
529
530                 cv::Mat distCoeffs(5, 1, cv::DataType<double>::type);
531                 distCoeffs.at<double>(0) = 0;//k1; // Because the function
                    solvePnP already undistort the imagePoints
532                 distCoeffs.at<double>(1) = 0;//k2;
533                 distCoeffs.at<double>(2) = 0;//p1;
534                 distCoeffs.at<double>(3) = 0; //p2;
```

```
535                 distCoeffs.at<double>(4) = 0; //k3;
536
537                 //std::cout << "World points" << std::endl << objectPoints <<
                        std::endl;
538                 //std::cout << "Image points" << std::endl << corner <<
                      std::endl;
539
540                 cv::Mat RotMat(3, 3, cv::DataType<double>::type);
541
542                 cv::solvePnP(objectPoints, corners2, cameraMatrix,
                      distCoeffs, rvec, tvec);// , false, CV_ITERATIVE);
543                 //std::cout << tvec << std::endl;
544                 //std::cout << rvec << std::endl;
545
546
547                 cv::Rodrigues(rvec, RotMat);
548                 LOG("RotMat", RotMat);
549                 X = tvec.at<double>(0);
550                 Y = tvec.at<double>(1);
551                 Z = tvec.at<double>(2);
552                 Xrob = X;
553                 Yrob = Y;
554                 Zrob = Z;
555
556                 cv::cv2eigen(RotMat, RotMatEigen);
557                 LOG("RotMatEigen", RotMatEigen);
558
559                 double pi = 3.141592;
560                 // H0
561                 // values for r0 and t0 gadderd from robot,
562                 //since it tacke picture from samme position every time this
                      is constant
563
564                 /*
565                 Rotation matrix from base to TCP
566                 Eular angels, roll, pitch and yaw
567                 rz = -90, ry = 0, rx -180
568                 */
569                 Matrix3d r0 = Matrix3d::Zero();
570                 double angle1 = azrob* pi / 180;//angel; // z-rotation
571                 double angle2 = ayrob* pi / 180;
572                 double angle3 = axrob* pi / 180;
573
574
575                 r0 = AngleAxisd(angle1, Vector3d::UnitZ())
576                     * AngleAxisd(angle2, Vector3d::UnitY())
577                     * AngleAxisd(angle3, Vector3d::UnitX());
578
579                 /*Position of TCP with respect to base of robot*/
580                 Vector3d t0;
581                 t0[0] = xrob;
582                 t0[1] = yrob;
583                 t0[2] = zrob;
584
585                 Matrix4d h0 = Matrix4d::Zero();
586                 h0(0, 0) = r0(0, 0); h0(0, 1) = r0(0, 1); h0(0, 2) = r0(0,
```

```
                    2);
587                 h0(1, 0) = r0(1, 0); h0(1, 1) = r0(1, 1); h0(1, 2) = r0(1,
                    2);
588                 h0(2, 0) = r0(2, 0); h0(2, 1) = r0(2, 1); h0(2, 2) = r0(2,
                    2);
589                 h0(0, 3) = t0(0); h0(1, 3) = t0(1); h0(2, 3) = t0(2); h0(3,
                    3) = 1;
590
591                 /* -----------h1 = [r1 t1]--------------
592                 From TCP to camera center
593                 Camera has samme rotation than TCP
594                 */
595                 Matrix3d r1 = Matrix3d::Identity();
596                 Vector3d t1;
597                 t1[0] = 0;//0;//10.02;//5.07; //2.42;               //
                    transpaltion of camera compered to TCP in Xaxis
598                 t1[1] = -160.2;//-196.7;//-161.5;//-196.9;//-200.2;//-147.5 -
                     50;//50;   // transpaltion of camera compered to TCP in
                    Yaxis
599                 t1[2] = -30;//-30.23;//     // transpaltion of camera
                    compered to TCP in Zaxis
600
601                 Matrix4d h1 = Matrix4d::Zero();
602                 h1(0, 0) = r1(0, 0); h1(0, 1) = r1(0, 1); h1(0, 2) = r1(0,
                    2);
603                 h1(1, 0) = r1(1, 0); h1(1, 1) = r1(1, 1); h1(1, 2) = r1(1,
                    2);
604                 h1(2, 0) = r1(2, 0); h1(2, 1) = r1(2, 1); h1(2, 2) = r1(2,
                    2);
605                 h1(0, 3) = t1(0); h1(1, 3) = t1(1); h1(2, 3) = t1(2); h1(3,
                    3) = 1;
606
607                 /* -----------h2 = [r2 t2]--------------
608                 From Camera frame to Plate position
609                 Only rotation about Z axis
610                 */
611
612                 Vector3d t2;
613                 t2[0] = X;  // x-pos of plate from camera frame
614                 t2[1] = Y;  //+centeroffset;        // y-pos of plate from
                    camera frame
615                 t2[2] = Z;  // z- pos of plate from camera frame
616
617
618                 Matrix4d h2 = Matrix4d::Identity();
619                 h2(0, 0) = RotMatEigen(0, 0); h2(0, 1) = RotMatEigen(0, 1);
                    h2(0, 2) = RotMatEigen(0, 2);
620                 h2(1, 0) = RotMatEigen(1, 0); h2(1, 1) = RotMatEigen(1, 1);
                    h2(1, 2) = RotMatEigen(1, 2);
621                 h2(2, 0) = RotMatEigen(2, 0); h2(2, 1) = RotMatEigen(2, 1);
                    h2(2, 2) = RotMatEigen(2, 2);
622                 h2(0, 3) = t2(0); h2(1, 3) = t2(1); h2(2, 3) = t2(2); h2(3,
                    3) = 1;
623                 LOG("h2", h2);
624
625                 //H4
```

```
626
627                 Matrix3d r4 = Matrix3d::Zero();
628                 Vector3d t4;
629                 if (boundingBox.size.width < boundingBox.size.height){
630                 r4(0, 0) = -1;
631                 r4(1, 1) = -1;
632                 r4(2, 2) = 1;
633                 LOG("r4", r4);
634                 }
635                 else {
636                     r4(0, 0) = 1;
637                     r4(1, 1) = -1;
638                     r4(2, 2) = -1;
639                     LOG("r4", r4);
640                 }
641
642
643                 t4[0] = plateoffsetx; //plateoffsetx;
644                 t4[1] = plateoffsety;   // plateoffsety;
645                 t4[2] = 0;  //0;
646
647                 Matrix4d h4 = Matrix4d::Zero();
648                 h4(0, 0) = r4(0, 0);
649                 h4(0, 1) = r4(0, 1);
650                 h4(0, 2) = r4(0, 2);
651
652                 h4(1, 0) = r4(1, 0);
653                 h4(1, 1) = r4(1, 1);
654                 h4(1, 2) = r4(1, 2);
655
656                 h4(2, 0) = r4(2, 0);
657                 h4(2, 1) = r4(2, 1);
658                 h4(2, 2) = r4(2, 2);
659
660                 h4(0, 3) = t4(0);
661                 h4(1, 3) = t4(1);
662                 h4(2, 3) = t4(2);
663                 h4(3, 3) = 1;
664
665                 LOG("h4", h4);
666
667
668                 /*------h3 = h0 * h1* h2 --------
669                 Representation of plate from base frame
670                 */
671                 Matrix4d h3;
672                 h3 = h0*h1*h2*h4;
673                 Matrix3d r3;
674                 r3(0, 0) = h3(0, 0); r3(0, 1) = h3(0, 1); r3(0, 2) = h3(0,    ↵
                      2);
675                 r3(1, 0) = h3(1, 0); r3(1, 1) = h3(1, 1); r3(1, 2) = h3(1,    ↵
                      2);
676                 r3(2, 0) = h3(2, 0); r3(2, 1) = h3(2, 1); r3(2, 2) = h3(2,    ↵
                      2);
677
678                 // Position of plate acording to base frame
```

```
679                 X = h3(0, 3);
680                 Y = h3(1, 3);
681                 Z = h3(2, 3);
682                 //cout << "h3" << h3 << endl;
683
684                 // Rotation of plate acording to base fram in Eular angels
685
686
687                 // Rotation matrix to eular angels
688                 //https://www.geometrictools.com/Documentation/
                      EulerAngles.pdf
689                 if (r3(2, 0) < 1)
690                 {
691                     if (r3(2, 0) > -1)
692                     {
693                         rotY = asin(-r3(2, 0));
694                         rotZ = atan2(r3(1, 0), r3(0, 0));
695                         rotX = atan2(r3(2, 1), r3(2, 2));
696                     }
697                     else // r 2 0 = -1
698                     {
699                         // Not a u n i q u e s o l u t i o n : t he t aX - t
                        h e t aZ = a t a n 2(-r12 , r 1 1 )
700                         rotY = pi / 2;
701                         rotZ = -atan2(-r3(1, 2), r3(1, 1));
702                         rotX = 0;
703                     }
704                 }
705                 else // r 2 0 = +1
706                 {
707                     // Not a u n i q u e s o l u t i o n : t he t aX + t h e
                    t aZ = a t a n 2(-r12 , r 1 1 )
708                     rotY = -p1 / 2;
709                     rotZ = atan2(-r3(1, 2), r3(1, 1));
710                     rotX = 0;
711                 }
712
713                 rotZ = rotZ * 180 / pi;
714                 rotY = rotY * 180 / pi;
715                 rotX = rotX * 180 / pi;
716
717                 // Project 3D points onto 2D image
718                 std::vector<cv::Point3f> p3d;
719                 std::vector<cv::Point2f> p2d;
720                 cv::Point3f p0, pX, sX, sY,  sZ;
721
722
723                 // Origo
724
725
726                 p0.x = 0;// hcpMatrix.at<double>(0, 3);
727                 p0.y = 0;// hcpMatrix.at<double>(1, 3);
728                 p0.z = 0;// hcpMatrix.at<double>(2, 3);
729
730                  // Local vector s
731                 sX.x = 100.0;
```

```
732                 sX.y = 0.0;
733                 sX.z = 0.0;
734
735                 sY.x = 0.0;
736                 sY.y = 100.0;
737                 sY.z = 0.0;
738
739                 sZ.x = 0.0;
740                 sZ.y = 0.0;
741                 sZ.z = 100.0;
742
743                 p3d.push_back(p0);
744                 p3d.push_back(sX);
745                 p3d.push_back(sY);
746                 p3d.push_back(sZ);
747
748                 cv::projectPoints(p3d, rvec, tvec, cameraMatrix, distCoeffs,
                      p2d);
749
750                 cv::arrowedLine(drawing, p2d[0], p2d[1], red, 5, 8); //xline
751                 cv::arrowedLine(drawing, p2d[0], p2d[2], green, 5, 8); //
                      yline
752                 cv::arrowedLine(drawing, p2d[0], p2d[3], blue, 5, 8); //zline
753
754                 Eigen::Matrix4d h5 = h2* h4;
755                 cv::Mat h5cv = cv::Mat(4, 4, CV_64F);
756                 cv::eigen2cv(h5, h5cv);
757                 LOG("h5cv", h5cv);
758
759                 rvecc = cv::Mat::zeros(cv::Size(3, 3), CV_64FC1);
760                 rvecc.at<double>(0, 0) = h5cv.at<double>(0, 0);
761                 rvecc.at<double>(0, 1) = h5cv.at<double>(0, 1);
762                 rvecc.at<double>(0, 2) = h5cv.at<double>(0, 2);
763
764                 rvecc.at<double>(1, 0) = h5cv.at<double>(1, 0);
765                 rvecc.at<double>(1, 1) = h5cv.at<double>(1, 1);
766                 rvecc.at<double>(1, 2) = h5cv.at<double>(1, 2);
767
768                 rvecc.at<double>(2, 0) = h5cv.at<double>(2, 0);
769                 rvecc.at<double>(2, 1) = h5cv.at<double>(2, 1);
770                 rvecc.at<double>(2, 2) = h5cv.at<double>(2, 2);
771                 LOG("rvecc", rvecc);
772
773                 tvecc.at<double>(0) = h5cv.at<double>(0, 3);//plateoffsetx;
774                 tvecc.at<double>(1) = h5cv.at<double>(1, 3);//plateoffsety;
775                 tvecc.at<double>(2) = h5cv.at<double>(2, 3);
776                 LOG("tvecc", tvecc);
777                 std::vector<cv::Point2f> p2dc;
778                 cv::projectPoints(p3d, rvecc, tvecc, cameraMatrix,
                      distCoeffs, p2dc);
779                 cv::arrowedLine(drawing, p2dc[0], p2dc[1], red, 5, 8); //
                      xline
780                 cv::arrowedLine(drawing, p2dc[0], p2dc[2], green, 5, 8); //
                      yline
781                 cv::arrowedLine(drawing, p2dc[0], p2dc[3], blue, 5, 8); //
                      zline
```

```
782
783                 t = ((double)getTickCount() - t) / getTickFrequency();
784                 LOGtime("Time for img prossesing = ");
785             }
786
787             //---------------------                                          ↵
                  UDP------------------------------------------------
788             std::cout << "" << std::endl;
789             udp();
790             zMeasured = zMeasured - 105-22; // differens to TCP and statonary ↵
                   diviation
791             std::cout << "Distance from tool to plate" << std::endl;
792             cout << "Measured: " << zMeasured << " mm" << endl;
793
794             t = ((double)getTickCount() - t) / getTickFrequency();
795             LOGtime("Time for reading Arduino sensor = ");
796
797                 // ------------------Position estemation                     ↵
                      -------------------------------------
798                 u = boundingBox.center.x;
799                 v = boundingBox.center.y;
800                 x = u - (imageSizeX / 2);
801                 y = v - (imageSizeY / 2);
802                 Z = zMeasured + 17 - 6 * (1 - imagecount + 1);  //953        ↵
                      nDatapop - imagecount + 1)
803                 X = (x / fx) *Z; // X pos of center of plate
804                 Y = (y / fy) *Z;    // Y pos of center of plate
805                 LOG("x  ", x);
806                 LOG("y  ", y);
807                 kinematics();
808             }
809
810             if (wallpos == 1) {
811                 Z = Z - 306;
812                 X = X + 3;
813                 std::cout << "Estimated: " << Yrob - Y << " mm"<< std::endl;
814             }
815             if (wallpos == 0) {
816                 std::cout << "Estimated: " << Zrob - Z << " mm" << std::endl;
817                 //if (Z > 90) Z = 85;
818                 //else if (Z < 80)  Z = 85;
819                 //else Z;
820
821             }
822
823             std::cout << "" << std::endl;
824             std::cout << "Angel of plate: " << angel << std::endl;
825
826             std::cout << "" << std::endl;
827             std::cout << "Position of plate relative to robot base" <<       ↵
                  std::endl;
828             nDataX = X;
829             nDataY = Y;
830             nDataZ = Z;
831             nDataAngelX = rotX;
832             nDataAngelY = rotY;
```

```
833             nDataAngelZ = rotZ;
834             nData = 0;
835             nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVarX,
                  sizeof(nDataX), &nDataX);
836             nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVarY,
                  sizeof(nDataY), &nDataY);
837             nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVarZ,
                  sizeof(nDataZ), &nDataZ);
838             nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_VALBYHND,
                  lHdlVarAngelX, sizeof(nDataAngelX), &nDataAngelX);
839             nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_VALBYHND,
                  lHdlVarAngelY, sizeof(nDataAngelY), &nDataAngelY);
840             nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_VALBYHND,
                  lHdlVarAngelZ, sizeof(nDataAngelZ), &nDataAngelZ);
841
842             nErr = AdsSyncWriteReq(pAddr, ADSIGRP_SYM_VALBYHND, lHdlVar,
                  sizeof(nData), &nData);
843             if (nErr) cerr << "Error: AdsSyncWriteReq: " << nErr << '\n';
844
845             LOG("u  ",u);
846             LOG("v  ", v);
847
848             LOG( "xrob ", xrob);
849             LOG( "yrob " ,yrob);
850             LOG( "zrob " ,zrob);
851             LOG( "axrob " ,axrob);
852             LOG( "ayrob " ,ayrob);
853             LOG( "azrob " ,azrob);
854
855             cout << "X " << X << endl;
856             cout << "Y " << Y << endl;
857             cout << "Z " << Z << endl;
858             cout << "rotZ " << rotZ << endl;
859             cout << "rotY " << rotY << endl;
860             cout << "rotX " << rotX << endl;
861
862             std::cout << "" << std::endl; //<< "Finished" << std::endl <<
                  ""<< std::endl;
863
864             foutsave << X << "," << Y << "," << Z << "," << rotX << "," <<
                  rotY << "," << rotZ << "," << angel << endl;
865
866
867             LOG("Corner 0: ",corners[0]);
868             LOG("Corner 1: ", corners[1]);
869             LOG("Corner 2: ", corners[2]);
870             LOG("Corner 3: ", corners[3]);
871             LOG("Rectangel width: ", boundingBox.size.width);
872             LOG("Rectangel hight: ", boundingBox.size.height);
873
874
875             if (imgShow) {
876
877                 //draw center of img
878                 //circle(drawing, Point(imageSizeX / 2, imageSizeY / 2), 4,
                      Scalar(255, 255, 255), -1, 8, 0);
```

```
879                 //putText(drawing, "center of img", Point(imageSizeX / 2 + 5,
                        imageSizeY / 2), FONT_HERSHEY_SIMPLEX, 1, Scalar(0, 200,
                      200), 4);
880                 circle(drawing, Point(center), 4, black, 8, 0);
881                 for (int i = 0; i < 4; i++) {
882                     line(drawing, corners[i], corners[(i + 1) % 4], black, 5,
                        CV_AA);
883                     stringstream Cornertext; Cornertext << "Corner: " << i;
884                     //putText(drawing,Cornertext.str(), (Point(corners[i])),
                       FONT_HERSHEY_SIMPLEX, 1, black, 4);
885                     waitKey(1);
886                     showimg("Draw img", drawing);
887                     showimg("Undistorted image", imgUndistorted);
888                     //showimg("Original img", img);
889                     cv::moveWindow("Draw img", 650, 440);
890                     cv::moveWindow("Undistorted image", 650, 0);
891                     waitKey(1);
892                 }
893
894
895             }
896             if (imageHold){
897
898
899                 while (true)
900                 {
901                     int c;
902                     c = waitKey(20);
903                     if ((char)c == 27)
904                     {
905                         break;
906                     }
907                 }
908
909
910             }
911         }
912
913     }
914         // Close the communication port
915         nErr = AdsPortClose();
916         foutsave.close();
917         if (nErr) cerr << "Error: AdsPortClose: " << nErr << '\n';
918
919         return exitCode;
920 }
921
922
923 void Threshold_Demo(int, void*)
924 {
925     /* 0: Binary
926     1: Binary Inverted
927     2: Threshold Truncated
928     3: Threshold to Zero
929     4: Threshold to Zero Inverted
930     */
```

```
931
932     Mat dst;
933     threshold(imgUndistorted, dst, threshold_value, max_BINARY_value,
          threshold_type);
934     imshow("Threshold Demo", dst);
935 }
936
937 void udp(void)
938 /* This function opens a udp connection to the arduino and reads the
      posistion 10 times and takes the average
939 This function is based on:
940 https://social.msdn.microsoft.com/Forums/en-US/
      f7402dfa-405c-4b6d-95d8-3d10530d8120/using-server-and-a-client-with-udp?
      forum=Vsexpressvc
941 and modified by:
942 Remi Askeland
943 */
944 {
945 #define SERVER "192.168.1.3" //ip address of udp server
946 #define BUFLEN 512  //Max length of buffer
947 #define PORT 5000   //The port on which to listen for incoming data
948
949     struct sockaddr_in si_other;
950     int s, slen = sizeof(si_other);
951     char buf[BUFLEN];
952     char message[BUFLEN];
953     float max = 0;
954     WSADATA wsa;
955     //Initialise winsock
956     //printf("\nInitialising Winsock...");
957     if (WSAStartup(MAKEWORD(2, 2), &wsa) != 0)
958     {
959         cout << "Failed. Error Code: " << WSAGetLastError() << endl;
960         exit(EXIT_FAILURE);
961     }
962     //printf("Initialised.\n");
963
964     //create socket
965     //printf("\nInitialising socket...");
966     if ((s = socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == SOCKET_ERROR)
967     {
968         printf("socket() failed with error code : %d", WSAGetLastError());
969         exit(EXIT_FAILURE);
970     }
971     //printf("Initialised.\n");
972
973     //setup address structure
974     memset((char *)&si_other, 0, sizeof(si_other));
975     si_other.sin_family = AF_INET;
976     si_other.sin_port = htons(PORT);
977     si_other.sin_addr.S_un.S_addr = inet_addr(SERVER);
978
979     //start communication
980     const int udpcounter = 10;
981     float i = 0;
982     double input[udpcounter];
```

```
983      int numOfInputs = 0;
984      double sum = 0;
985      while (i<udpcounter)
986      {
987          //send the message
988          if (sendto(s, "zpos", strlen(message), 0, (struct sockaddr *)
               &si_other, slen) == SOCKET_ERROR)
989          {
990              printf("sendto() failed with error code : %d", WSAGetLastError
                   ());
991              exit(EXIT_FAILURE);
992          }
993
994          //receive a reply and print it
995          //clear the buffer by filling null, it might have previously received
               data
996          memset(buf, '\0', BUFLEN);
997          //try to receive some data, this is a blocking call
998          if (recvfrom(s, buf, BUFLEN, 0, (struct sockaddr *) &si_other, &slen)
               == SOCKET_ERROR)
999          {
1000             printf("recvfrom() failed with error code : %d", WSAGetLastError
                   ());
1001             exit(EXIT_FAILURE);
1002         }
1003
1004
1005         max = atof(buf);
1006
1007             //puts(buf);
1008             //cout << buf << endl;
1009             input[numOfInputs] = atof(buf);
1010             sum += input[numOfInputs];
1011             numOfInputs++;
1012             i = i + 1;
1013     }
1014     zMeasured = sum / numOfInputs;
1015     //cout << "The sum is " << sum << endl;
1016     //cout << "Number of inputs " << numOfInputs << endl;
1017     //cout << "Average of zinputs " << zMeasured << endl;
1018     closesocket(s);
1019     WSACleanup();
1020
1021 }
1022
1023 void showimg(string windowName, Mat imgVariabel) {
1024
1025     namedWindow(windowName, CV_WINDOW_NORMAL);
1026     resizeWindow(windowName, 600, 400);
1027     imshow(windowName, imgVariabel);
1028     waitKey(1);
1029 }
1030
```

```
MODULE platehandler_h2017
    ! TOOl and objects
    TASK PERS tooldata Bachelor_data_modifyed:=[TRUE,[[60.863,5.254,292.718505896],
[0.661754661,-0.283826324,0.26264135,-0.642295032]],[1,[0,0,1],[1,0,0,0],0,0,0]];
    TASK PERS wobjdata Small_pallet:=[FALSE,TRUE,"",[[593.35,-1260,70.5],
[0,-0.707106781,0.707106781,0]],[[0,0,0],[1,0,0,0]]];
    TASK PERS tooldata Tool:=[TRUE,[[51.632026529,-0.940509669,197.87948077],
[0.382684336,-0.00000314,-0.923879158,-0.000002319]],[1,[0,0,1],[1,0,0,0],0,0,0]];
    TASK PERS tooldata Calibrated_data:=[TRUE,[[152.449,-5.788,144.658],
[0.499889488,0.714129269,-0.489899498,-0.011329988]],[1,[0,0,1],[1,0,0,0],0,0,0]];
    TASK PERS tooldata Bachelor_data:=[TRUE,[[60.863,5.254,292.718505896],
[0.922102381,-0.014980039,0.386410998,0.013760036]],[1,[0,0,1],[1,0,0,0],0,0,0]];
    TASK PERS wobjdata Big_pallet:=[FALSE,TRUE,"",[[1660.85,-1255,89.5],
[0,-0.707106781,0.707106781,0]],[[0,0,0],[1,0,0,0]]];
    TASK PERS tooldata Tooldata_1711:=[TRUE,[[175.571,192.0081,171.4725],
[0.484496568,0.713500625,-0.505914514,-0.015180221]],[10,[0,0,1],[1,0,0,0],0,0,0]];
    TASK PERS tooldata Tooldata_1711withorientation:=[TRUE,
[[85.402642691,4.133520763,236.967231212],[0.661754661,-0.283826324,0.26264135,-0.642295032]],[1,
[0,0,1],[1,0,0,0],0,0,0]];
    TASK PERS tooldata Tooldata_2011:=[TRUE,[[121.626434098,-18.142853011,297.675631922],
[0.665475549,-0.274988922,0.25406496,-0.645735536]],[10,[0,0,1],[1,0,0,0],0,0,0]];
    TASK PERS wobjdata Turntool:=[FALSE,TRUE,"",[[1705.85,-141,1374.5],
[0,-0.707106781,0.707106781,0]],[[0,0,0],[1,0,0,0]]];
    TASK PERS wobjdata Wall:=[FALSE,TRUE,"",[[245.85,1400,2167.5],[0.707106781,-0.707106781,0,0]],
[[0,0,0],[1,0,0,0]]];
    TASK PERS tooldata Tool2911_modrot:=[TRUE,[[55.083309317,4.806686265,268.901010755],
[0.665475549,-0.274988922,0.25406496,-0.645735536]],[10,[0,0,1],[1,0,0,0],0,0,0]];
    ! Targets
    CONST robtarget homehome:=[[1547.954,-1028.131,1809.932],[0,-0.707106781,0.707106781,0],
[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p1:=[[0,0,-1500],[1,0,0,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p2:=[[0,0,-1000],[1,0,0,0],[-1,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p3:=[[0,157.5,-1000],[1,0,0,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E
+09]];
    CONST robtarget p4:=[[0,157.5,-857.5],[1,0,0,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E
+09]];
    CONST robtarget p5:=[[0,0,-500],[1,0,0,0],[-1,0,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p6:=[[0,500,-500],[1,0,0,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p7:=[[0,0,-500],[1,0,0,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p8:=[[0,0,-200],[1,0,0,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p9:=[[0,0,0],[1,0,0,0],[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p10:=[[0,500,-500],[0,0,1,0],[-1,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p11:=[[0,500,-200],[0,0,1,0],[-1,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p12:=[[0,500,0],[0,0,1,0],[-1,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p13:=[[0,0,200],[0,0,1,0],[-1,1,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p14:=[[0,0,6],[0,0,1,0],[-1,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p15:=[[0.000048162,-0.000046612,-133.00000186],
[0.000000037,0.000000071,1,0.000000057],[-1,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p16:=[[0,0,-500],[0,0,1,0],[-1,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p17:=[[0,500,-500],[0,0,1,0],[-1,1,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget p18:=[[0,500,-500],[0.707106781,0,-0.707106781,0],[-1,1,-1,0],[9E+09,9E+09,9E
+09,9E+09,9E+09,9E+09]];
    CONST robtarget wall1:=[[300,150,-700],[1,0,0,0],[0,1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E
+09]];
    CONST robtarget pSmallBottom:=[[0,0,-6],[1,0,0,0],[-1,1,-3,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E
+09]];
    CONST robtarget pSmallBottom_2:=[[0,0,-12],[1,0,0,0],[-1,1,-3,0],[9E+09,9E+09,9E+09,9E+09,9E
+09,9E+09]];
```

```
    CONST robtarget pSmallBottom_3:=[[0,0,-18],[1,0,0,0],[-1,1,-3,0],[9E+09,9E+09,9E+09,9E+09,9E
+09,9E+09]];
    CONST robtarget pSmallBottom_4:=[[0,0,-24],[1,0,0,0],[-1,1,-3,0],[9E+09,9E+09,9E+09,9E+09,9E
+09,9E+09]];
    CONST robtarget pSmallBottom_5:=[[0,0,-30],[1,0,0,0],[-1,1,-3,0],[9E+09,9E+09,9E+09,9E+09,9E
+09,9E+09]];
    CONST robtarget pSmallBottom_6:=[[0,0,-36],[1,0,0,0],[-1,1,-3,0],[9E+09,9E+09,9E+09,9E+09,9E
+09,9E+09]];
    CONST robtarget pSmallBottom_7:=[[0,0,-42],[1,0,0,0],[-1,1,-3,0],[9E+09,9E+09,9E+09,9E+09,9E
+09,9E+09]];
    CONST robtarget pSmallBottom_8:=[[0,0,-48],[1,0,0,0],[-1,1,-3,0],[9E+09,9E+09,9E+09,9E+09,9E
+09,9E+09]];
    CONST robtarget pSmallBottom_9:=[[0,0,-54],[1,0,0,0],[-1,1,-3,0],[9E+09,9E+09,9E+09,9E+09,9E
+09,9E+09]];
    CONST robtarget pSmallBottom_10:=[[0,0,-60],[1,0,0,0],[-1,1,-3,0],[9E+09,9E+09,9E+09,9E+09,9E
+09,9E+09]];
    CONST robtarget towall:=[[1047.954,-528.131,1809.932],[0,-0.707106781,0.707106781,0],
[-1,0,-1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget towall2_2:=[[1147.954,371.869,1809.932],[0,0,0.707106781,0.707106781],
[-1,0,0,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget towallpic1:=[[520,577.5,1942.7],[0,0,0.707106781,0.707106781],[0,0,1,0],[9E
+09,9E+09,9E+09,9E+09,9E+09,9E+09]];


    ! Sockets definition
    VAR socketdev tempSocket;
    VAR socketdev clientSocket;
    VAR string receivedString;

    !receiveCoordinates variables
        !Posistion in the string
        VAR num xSpos;
        VAR num ySpos;
        VAR num zSpos;
        VAR num angleXSpos;
        VAR num angleYSpos;
        VAR num angleZSpos;
        VAR num turnSpos;
        VAR num numbPlateSpos;
        ! Variables for reading coordinates
        VAR String xNew;
        VAR String yNew;
        VAR String zNew;
        VAR String angleXNew;
        VAR String angleYNew;
        VAR String angleZNew;
        VAR String turnNew;
        VAR String numbPlateNew;
        ! String to number conversation variables
        VAR bool    xBool;
        VAR num     xPlate;
        VAR bool    yBool;
        Var num     yPlate;
        VAR bool    zBool;
        VAR num     zPlate;
        VAR bool    angelXBool;
        VAR num     angleXPlate;
        VAR bool    angelYBool;
```

```
        VAR num     angleYPlate;
        VAR bool    angelZBool;
        VAR num     angleZPlate;
        VAR bool    turnBool;
        VAR num     turnPlateVar;
        VAR bool    numbPlateBool;
        VAR num     numbPlate;
        VAR orient    plateOrient;

        Var robtarget MyPosition;
        Var num myX;
        Var num myY;
        Var num myZ;
        Var num angleX;
        Var num angleY;
        Var num angleZ;
        VAR String sendCoord;

        VAR bool WallposBool;


    TASK PERS tooldata tool2:=[TRUE,[[175.571,192.008,171.473],
[0.484497,0.713501,-0.505915,-0.0151802]],[10,[0,0,0],[1,0,0,0],0,0,0]];
    TASK PERS tooldata tool2911:=[TRUE,[[168.438,7.1398,159.578],[1,0,0,0]],[1,[0,0,0],
[1,0,0,0],0,0,0]];




    PROC platehandler_main ()
        !=================initialize=================================
        DeActUnit M7DM1;            ! disconests the external axis
        TPErase;                    ! used to clear the display of the FlexPendant.
        TCPCreateConnection;        ! Connects the TCP/IP connection
        TPWrite "Connected to Beckhoff CX2040";
        WaitTime (1);
        !=====================Main-program============================


    WHILE NOT receivedString = "Stop" DO
    MsgReceive ;                ! Wait for message from client

    !Picture posistion small pallet
    IF receivedString = "Home" THEN
        SocketSend clientSocket\Str:=receivedString;
        home1;
        WaitTime (1);
        SocketSend clientSocket\Str:=receivedString;
        receivedString:="";
    ENDIF


    !Picture posistion small pallet
    IF receivedString = "Go to small pallet" THEN
        SocketSend clientSocket\Str:=receivedString;
        hometosmall;
        WaitTime (1);
```

```
        SocketSend clientSocket\Str:=receivedString;
        receivedString:="";
    ENDIF

    !Picture posistion 2 big pallet
    IF receivedString = "Go to big pallet" THEN
        SocketSend clientSocket\Str:=receivedString;
        hometobig;
        WaitTime (1);
        SocketSend clientSocket\Str:=receivedString;
        WallposBool := FALSE;
        receivedString:="";
    ENDIF

    !Pick plate
    IF receivedString = "Pick plate" THEN
        SocketSend clientSocket\Str:=receivedString;
        pickPlate;
        WaitTime (1);
        SocketSend clientSocket\Str:=receivedString;
        receivedString:="";
    ENDIF

      !Wall
    IF receivedString = "Wall" THEN
        SocketSend clientSocket\Str:=receivedString;
        toWall1;
        WaitTime (1);
        SocketSend clientSocket\Str:=receivedString;
        WallposBool := TRUE;
        receivedString:="";
    ENDIF

    !Wall Away
    IF receivedString = "Wall away" THEN
        SocketSend clientSocket\Str:=receivedString;
        MoveL [[xPlate,yPlate-50,zPlate],plateOrient,[0,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E
+09]],v20,fine,Tool2911_modrot\WObj:=wobj0;
        WaitTime (1);
        SocketSend clientSocket\Str:=receivedString;
        receivedString:="";
    ENDIF


    ! Home to turn, place plate on turn tool
    IF receivedString = "Turn1" THEN
        SocketSend clientSocket\Str:="Turn1";
        hometoturn;
        WaitTime (1);
        receivedString:="";
        SocketSend clientSocket\Str:="Turn1";
    ENDIF

    ! Dismount from plate and turns tool
    IF receivedString = "Turn2" THEN
        SocketSend clientSocket\Str:="Turn2";
        turntoturn;
        WaitTime (1);
```

```
    receivedString:="";
    SocketSend clientSocket\Str:="Turn2";
ENDIF

! Dismount from plate and turns tool
IF receivedString = "Turn3" THEN
    SocketSend clientSocket\Str:="Turn3";
    turntoturn2;
    WaitTime (1);
    receivedString:="";
    SocketSend clientSocket\Str:="Turn3";
ENDIF


! Coordinats fom client
IF receivedString = "Coordinates" THEN
    SocketSend clientSocket\Str:="Coordinates";
    WaitTime(1);
    SocketSend clientSocket\Str:="Coordinates";
    receiveCoordinates;
    WaitTime (1);
    SocketSend clientSocket\Str:=receivedString;
    WaitTime (1);
    SocketSend clientSocket\Str:=receivedString;
    receivedString:="";
ENDIF

IF receivedString = "Robot position" THEN
    SocketSend clientSocket\Str:=receivedString;
    WaitTime(1);
    sendcoordinats;
    WaitTime(1);
    receivedString:="";
ENDIF


ENDWHILE

! Stop sequence
WHILE receivedString = "Stop" DO
    SocketSend clientSocket\Str:="Stop";
    StopMove ;
    SocketSend clientSocket\Str:="Stop";
    TCPShutdown ;
    WaitTime (1);
    receivedString:="";
    RETURN ;
ENDWHILE
!====================Main-END=============================================
ENDPROC


PROC TCPCreateConnection()
    ! setup socket
    SocketCreate tempSocket;
    SocketBind tempSocket,"192.168.1.4",5000;
    SocketListen tempSocket;
```

```
    ! Waiting for a connection request
    SocketAccept tempSocket,clientSocket\Time:=WAIT_MAX;

    ! Communication
    !SocketSend clientSocket\Str:="Connected";


ENDPROC

 PROC msgReceive()
    receivedString:="";
    SocketReceive clientSocket\Str:=receivedString\Time:=WAIT_MAX;
    TPWrite receivedString;

ENDPROC

PROC TCPShutdown()
    ! Close sockets
    TPWrite "Client wrote - "+receivedString;
    receivedString:="";
    SocketSend clientSocket\Str:="Shutdown";
    WaitTime 5;
    SocketClose clientSocket;
    SocketClose tempSocket;
ENDPROC

PROC TCPManualShutdown()
    SocketClose clientSocket;
    SocketClose tempSocket;
ENDPROC

PROC receiveCoordinates()

    receivedString:="";
    SocketReceive clientSocket\Str:=receivedString\Time:=WAIT_MAX;
    !Find start and endpoint in xyz-string
    xSpos:=StrFind(receivedString,1,"x");
    ySpos:=StrFind(receivedString,1,"y");
    zSpos:=StrFind(receivedString,1,"z");
    angleXSpos:=StrFind(receivedString,1,"a");
    angleYSpos:=StrFind(receivedString,1,"b");
    angleZSpos:=StrFind(receivedString,1,"c");
    turnSpos:=StrFind(receivedString,1,"t");
    numbPlateSpos:=StrFind(receivedString,1,"n");
    !found_ack:=StrFind(receivedString,1,"q");

    !Finds xyz in string
    xNew:=strPart(receivedString,(xSpos+1),(ySpos-xSpos-1));
    yNew:=strPart(receivedString,(ySpos+1),(zSpos-ySpos-1));
    zNew:=strPart(receivedString,(zSpos+1),(angleXSpos-zSpos-1));
    angleXNew:=strPart(receivedString,(angleXSpos+1),(angleYSpos-angleXSpos-1));
    angleYNew:=strPart(receivedString,(angleYSpos+1),(angleZSpos-angleYSpos-1));
    angleZNew:=strPart(receivedString,(angleZSpos+1),(turnSpos-angleZSpos-1));
    turnNew:=strpart(receivedString,(turnSpos+1),(numbPlateSpos-turnSpos-1));
    numbPlateNew:=strpart(receivedString,(numbPlateSpos+1),1);
    !ack:=strpart(received_string,(found_ack+1),1);

    !convets string to num
```

```
        xBool:=StrToVal(xnew,xPlate);
        TPWrite "X= "\Num:=xPlate;
        yBool:=StrToVal(yNew,yPlate);
        TPWrite "Y= "\Num:=yPlate;
        zBool:=StrToVal(znew,zPlate);
        TPWrite "Z= "\Num:=zPlate;

        angelZBool:=StrToVal(angleZNew,angleZPlate);
        TPWrite "AngleZ= "\Num:=angleZPlate;
        angelYBool:=StrToVal(angleYNew,angleYPlate);
        TPWrite "AngleY= "\Num:=angleYPlate;
        angelXBool:=StrToVal(angleXNew,angleXPlate);
        TPWrite "AngleX= "\Num:=angleXPlate;


        turnBool:=StrToVal(turnNew,turnPlateVar);
        TPWrite "Turn= "\Num:=turnPlateVar;
        numbPlateBool:=StrToVal(numbPlateNew,numbPlate);
        TPWrite "Number of plates= "\Num:=numbPlate;
        !ok_ack:=StrToVal(ack,plate_ack);
        !TPWrite "ack = "\Num:=plate_ack;

        !z_distance := Cpos (\Tool:=NewToolpoint\WObj:=Pallet);
        !z_distance:=410-plate_z;
        !orientation of plate
        plateOrient:=OrientZYX(angleZPlate,angleYPlate,angleXPlate);
        !received_string:="";

    ENDPROC

    PROC pickPlate()
        DeActUnit M7DM1;
        IF WallposBool = FALSE THEN
            MoveL [[xPlate,yPlate,zPlate+100],plateOrient,[-1,1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E
+09,9E+09]],v200,fine,Tool2911_modrot\WObj:=wobj0;
            MoveL [[xPlate,yPlate,zPlate],plateOrient,[-1,1,-2,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E
+09]], v20, fine, Tool2911_modrot\WObj:=wobj0;
        ELSEIF WallposBool = True THEN
            MoveL [[xPlate,yPlate-50,zPlate],plateOrient,[0,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E
+09]],v200,fine,Tool2911_modrot\WObj:=wobj0;
            MoveL [[xPlate,yPlate,zPlate],plateOrient,[0,0,1,0],[9E+09,9E+09,9E+09,9E+09,9E+09,9E
+09]],v20,fine,Tool2911_modrot\WObj:=wobj0;
        ENDIF

    ENDPROC

    ! Kan slettes seiner
   PROC home1()
        DeActUnit M7DM1;
        MoveJ homehome,v500,fine,Tool2911_modrot\WObj:=wobj0;
    ENDPROC
    PROC hometosmall()
        DeActUnit M7DM1;
        !MoveJ homehome,v100,fine,Tooldata_2011\WObj:=wobj0;
        !MoveL p1,v100,fine,Tooldata_2011\WObj:=Small_pallet;
        MoveJ p2,v500,fine,Tool2911_modrot\WObj:=Small_pallet;
        IF numbPlate = 1 THEN
            MoveL pSmallBottom,v200,fine,Tool2911_modrot\WObj:=Small_pallet;
```

```
        ELSEIF numbPlate = 2 THEN
            MoveL pSmallBottom_2,v200,fine,Tool2911_modrot\WObj:=Small_pallet;
        ELSEIF  numbPlate = 3 THEN
            MoveL pSmallBottom_3,v200,fine,Tool2911_modrot\WObj:=Small_pallet;
         ELSEIF  numbPlate = 4 THEN
            MoveL pSmallBottom_4,v200,fine,Tool2911_modrot\WObj:=Small_pallet;
         ELSEIF  numbPlate = 5 THEN
            MoveL pSmallBottom_5,v200,fine,Tool2911_modrot\WObj:=Small_pallet;
         ELSEIF  numbPlate = 6 THEN
            MoveL pSmallBottom_6,v200,fine,Tool2911_modrot\WObj:=Small_pallet;
        ELSEIF  numbPlate = 7 THEN
            MoveL pSmallBottom_7,v200,fine,Tool2911_modrot\WObj:=Small_pallet;
        ELSEIF  numbPlate = 8 THEN
            MoveL pSmallBottom_8,v200,fine,Tool2911_modrot\WObj:=Small_pallet;
        ELSEIF  numbPlate = 9 THEN
            MoveL pSmallBottom_9,v200,fine,Tool2911_modrot\WObj:=Small_pallet;
        ELSEIF  numbPlate = 10 THEN
            MoveL pSmallBottom_10,v200,fine,Tool2911_modrot\WObj:=Small_pallet;
        ENDIF

    ENDPROC

    PROC sendcoordinats()
        MyPosition := CRobT(\Tool:=Tool2911_modrot\Wobj:=wobj0);
        myX := MyPosition.trans.x;
        myY := MyPosition.trans.y;
        myZ := MyPosition.trans.z;
        angleX := EulerZYX(\X, MyPosition.rot);
        angleY := EulerZYX(\Y, MyPosition.rot);
        angleZ := EulerZYX(\Z, MyPosition.rot);
        TPWrite "My xPos= "\Num:=myX;
        TPWrite "My yPos= "\Num:=myY;
        TPWrite "My zPos= "\Num:=myZ;
        TPWrite "My zAngle= "\Num:=angleZ;
        TPWrite "My yAngle= "\Num:=angleY;
        TPWrite "My xAngle= "\Num:=angleX;
        sendCoord :="x"+ NumToStr(myX,4)+ "y"+ NumToStr(myY,4)+"z"+NumToStr(myZ,4)+"ax"+NumToStr
(angleX,4)+"ay"+NumToStr(angleY,4)+"az"+NumToStr(angleZ,4) ;
        !TPWrite "sendCoord= "+ sendCoord;
        SocketSend clientSocket\Str:=sendCoord;
    ENDPROC



    PROC hometobig()
        DeActUnit M7DM1;
        !MoveJ homehome,v100,fine,Tooldata_2011\WObj:=wobj0;
        MoveJ p3,v500,fine,Tool2911_modrot\WObj:=Big_pallet;
        MoveL p4,v300,fine,Tool2911_modrot\WObj:=Big_pallet;
        !MoveL p5,v100,fine,Tooldata_2011\WObj:=Big_pallet;
    ENDPROC
    PROC hometoturn()
        DeActUnit M7DM1;
        MoveJ homehome,v100,fine,Tooldata_2011\WObj:=wobj0;
        MoveL p6,v100,fine,Tooldata_2011\WObj:=Turntool;
        MoveL p7,v100,fine,Tooldata_2011\WObj:=Turntool;
        MoveL p8,v100,fine,Tooldata_2011\WObj:=Turntool;
        MoveL p9,v100,fine,Tooldata_2011\WObj:=Turntool;
```

```
    ENDPROC

    PROC turntoturn()
        DeActUnit M7DM1;
        MoveL p8,v100,fine,Tooldata_2011\WObj:=Turntool;
        MoveL p7,v100,fine,Tooldata_2011\WObj:=Turntool;
        MoveL p6,v100,fine,Tooldata_2011\WObj:=Turntool;
        MoveJ p10,v100,fine,Tooldata_2011\WObj:=Turntool;
        MoveL p11,v100,fine,Tooldata_2011\WObj:=Turntool;
        MoveL p12,v100,fine,Tooldata_2011\WObj:=Turntool;
        MoveL p13,v100,fine,Tooldata_2011\WObj:=Turntool;
        MoveL p14,v100,fine,Tooldata_2011\WObj:=Turntool;
        MoveL p15,v100,fine,Tooldata_2011\WObj:=Turntool;
    ENDPROC
    PROC turntoturn2()
        DeActUnit M7DM1;
        MoveL p16,v100,fine,Tooldata_2011\WObj:=Turntool;
        MoveL p17,v100,fine,Tooldata_2011\WObj:=Turntool;
        MoveJ wall1,v100,fine,Tooldata_2011\WObj:=Wall;
    ENDPROC

    PROC toWall1()
        DeActUnit M7DM1;
        MoveJ towall,v500,fine,Tool2911_modrot\WObj:=wobj0;
        MoveJ towall2_2,v500,fine,Tool2911_modrot\WObj:=wobj0;
         MoveL towallpic1,v500,fine,Tool2911_modrot\WObj:=wobj0;
    ENDPROC

ENDMODULE
```

Arduino kode

Avstandsmåling med VL53L0X sensor
Kommunikasjon med UDP

```
#include <Ethernet.h> //Load Ethernet Library
#include <EthernetUdp.h> //Load UDP Library
#include <SPI.h> //Load the SPI Library

byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xEE}; //Assign a mac address
IPAddress ip(192, 168, 1, 3); //Assign my IP adress
unsigned int localPort = 5000; //Assign a Port to talk over
char packetBuffer[UDP_TX_PACKET_MAX_SIZE];
String datReq; //String for our data
int packetSize; //Size of Packet
EthernetUDP Udp; //Define UDP Object

#include <Wire.h>
#include <VL53L0X.h>
VL53L0X sensor;

void setup() {

  Serial.begin(9600); //Turn on Serial Port
  Ethernet.begin(mac, ip); //Initialize Ethernet
  Udp.begin(localPort); //Initialize Udp
  delay(1500); //delay
  Wire.begin();
  sensor.init();
  sensor.setTimeout(500);
  // Start continuous back-to-back mode (take readings as fast as possible).  To use
  // continuous timed mode instead, provide a desired inter-measurement period in
  // ms (e.g. sensor.startContinuous(100)).
  sensor.startContinuous();
}
void loop() {
  packetSize = Udp.parsePacket(); //Read theh packetSize

  if (packetSize > 0) { //Check to see if a request is present

    Udp.read(packetBuffer, UDP_TX_PACKET_MAX_SIZE); //Reading the data request on the Udp
    String datReq(packetBuffer); //Convert packetBuffer array to string datReq

    if (datReq == "zpos") { //See if Red was requested

      Udp.beginPacket(Udp.remoteIP(), Udp.remotePort());  //Initialize Packet send
      Udp.print(sensor.readRangeContinuousMillimeters()); //Send string back to client
      if (sensor.timeoutOccurred()) {
        Udp.print(" TIMEOUT");
      }
      Udp.endPacket(); //Packet has been sent
    }
  }
  memset(packetBuffer, 0, UDP_TX_PACKET_MAX_SIZE);
}
```