Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Kybernetikk	Vår semesteret, 2014 Åpen
Forfatter: Martin Tykhelle	..... (signatur forfatter)
Fagansvarlig: Karl Skretting Veileder(e): Karl Skretting, Ståle Freyer	
Tittel på masteroppgaven: Identifisere objekter på et bord ved hjelp av bildebehandling i Matlab slik at de kan plukkes opp av robot. Engelsk tittel: Identifying objects on a table using image processing in Matlab so that they can be picked up by a robot.	
Studiepoeng: 30	
Emneord: Robot, ABB, Matlab, Duplo, Maskinsyn, Bildebehandling, Fargesegmentering, Hjørnedeteksjon	Sidetall: 64 + vedlegg/annet: 27 Stavanger, 12. Juni 2014 dato/år

Identifisere objekter på et bord ved hjelp av bildebehandling i Matlab slik at de kan plukkes opp av robot.

Martin Tykhelle

12. juni 2014

Forord

Denne oppgaven markerer slutten på mitt masterstudie innen kybernetikk. Det har vært to spennende og utfordrende år på Universitetet i Stavanger.

Takk til Karl Skretting og Ståle Freyer for god veiledning og tilbakemeldinger gjennom hele oppgaveprosessen.

Jeg vil også takke gjengen på lesesalen for en kjekk studietid og min familie for støtten jeg har fått.

Martin Tykhelle 11.06.2014

Sammendrag

Denne masteroppgaven har som mål å detektere DUPLO klosser ved hjelp av et kamera og bildebehandling gjennom MATLAB for så å plukke opp klossene med en ABB IRB140 robot. Resultatet er et program som kan brukes til demonstrasjon av roboten og bildebehandling, samt en rekke funksjoner som kan brukes videre i andre prosjekter eller undervisning.

Oppgaven bruker MATLAB til bildebehandling, posisjonene til de detekterte klossene blir så sendt til Rapid som utfører bevegelsene til roboten. Klossene blir detektert ved hjelp av fargesegmentering og orientering blir funnet med hjørnedeteksjon. For å finne punkter i rommet basert på punkter i bildet brukes geometrisk kamerakalibrering. Ved å bruke et A4 ark som referanse, dannes et koordinatsystem i Rapid som brukes for å posisjonere roboten over hver kloss.

Kommunikasjon med roboten bruker utviklingspakken PC SDK utviklet av ABB. Fargesegmenteringen bruker distansene L*a*b* fargerommet for å klassifisere de forskjellige fargene. Hjørnedeteksjonen bruker distansen fra sentrum av objektet for å finne hjørnene. Det er også gjort sammenligninger for tidsforbruk i de forskjellige delene av programmet. Selve deteksjonen bruker rundt 6-12 sekunder avhengig av størrelse på bilde og prosesseringsevne.

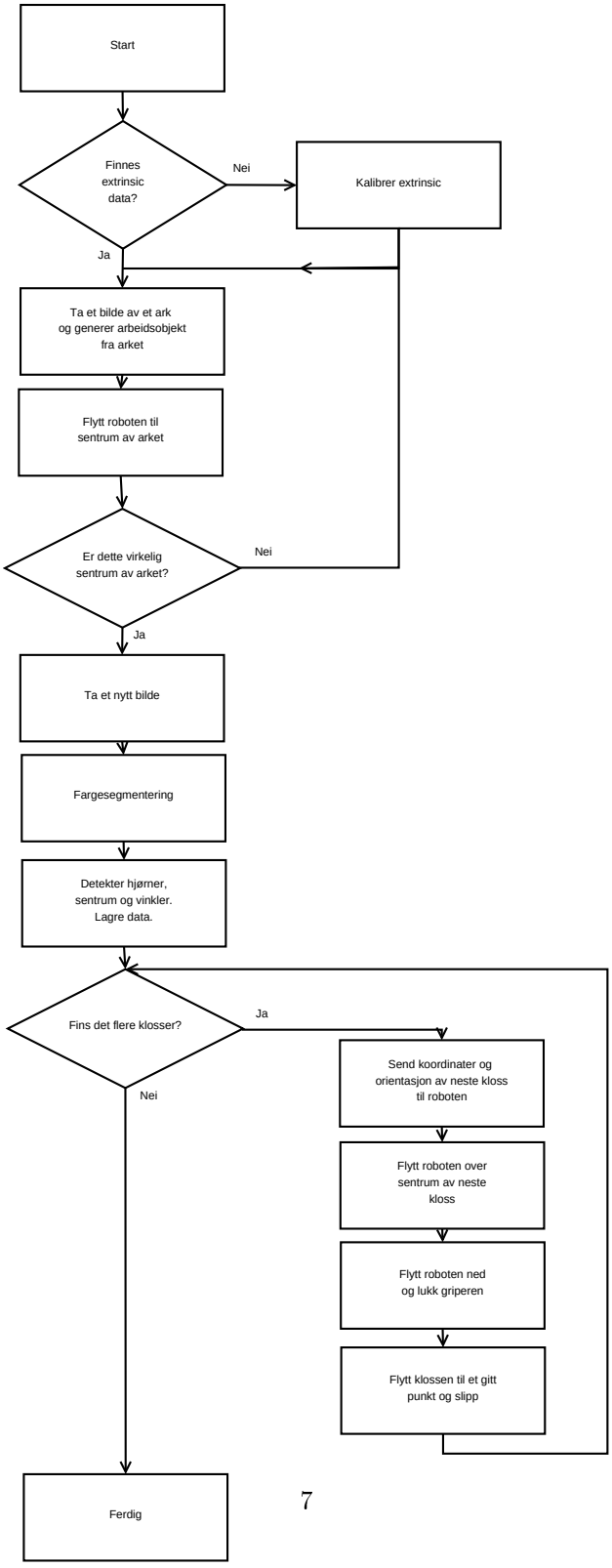
Innhold

Figurer

Tabeller

1 Introduksjon

I de siste 40 årene har fremskrittene innenfor robotteknologi og maskinsyn vært enorme. Industrielle roboter har gått fra å være klossete og skite maskiner til å bli presise verktøy. Universitetet i Stavanger sin robot-lab har to roboter av typen ABB IRB140, de er raske og små og utstyrt med gripeklør og penner. Til nå har robotene vært statiske, uten tilbakemelding fra virkeligheten. Et ønske fra instituttet er å lage et system som kan bruke et kamera for å gjenkjenne DUPLO klosser for så å plukke de opp med roboten. Denne løsningen skal kunne brukes som en demonstrasjon av roboten samtidig som det skal kunne brukes i videre arbeid.



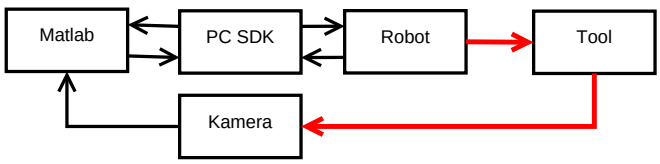
Figur 1: Skisse av løsningen.

Gjenkjenningen skal gjøres i MATLAB som skal sende koordinatene av de forskjellige klossene til roboten, som er konfigurert som en slave til MATLAB. Roboten flytter seg så til klossene og plukker de opp. De forskjellige fargene skal kunne skilles og nøyaktigheten på deteksjonen og plassering må være stor nok til å kunne stable klossene. Klossene skal plasseres på et papirark manuelt og plukkes opp derfra med roboten, posisjonen til klossene skal kunne finnes hver gang så lenge de ligger innenfor arket. Gjennom en full kjøring av programmet skal klossene være plukket og sortert/stablet.

For å løse dette blir et bilde fra kameraet kan brukes for å finne distansen i verden mellom to punkter i bildet. Før dette kan gjøres må indre og ytre (intrinsic og extrinsic) parametre kjennes. Disse finnes ved hjelp av Geometric Camera Calibration toolbox i MATLAB2013b (og nyere). MATLAB programmet skal kunne finne distansen fra et kjent punkt til et tilfeldig punkt i bildet og dele denne distansen opp i en X og en Y komponent. Dette skal brukes for å lage et arbeidsobjekt ut ifra et ark der origo i koordinatsystemet er det ene hjørnet av arket.

Ved å bruke hjørnet av arket som referanse kan posisjonen til klosser finnes i koordinatsystemet til det genererte arbeidsobjektet. Dersom en kloss for eksempel ligger 30 mm inn i arket fra hver side av arket blir koordinatene til klossen (30, 30). Først må klossene finnes ved hjelp av fargesegmentering og deteksjon av kanter, hjørner, sentrum og vinkler. Fargesegmenteringen deler bildet inn i binære bilder som inneholder klosser av en gitt farge. I hvert av disse bildene skal egenskapene til klossene finnes.

Roboten bruker et eget programmeringsspråk, Rapid, dette språket er utviklet av ABB. I Rapid kan posisjoner i rommet defineres som koordinater og orientering i forhold til et eksisterende arbeidsobjekt. Ved å benytte arket og arbeidsobjektet generert fra arket blir posisjonen av klossene funnet i virkeligheten. Denne informasjonen blir sendt til roboten som plukker opp klossene og plasserer de på et definert sted avhengig av form eller farge. Dette krever at MATLAB og Rapid kan kommunisere sammen, i tidligere prosjekter og undervisning har ABB sin løsning PC SDK blitt brukt til dette.



Figur 2: Sammenheng mellom de forskjellige fysiske elementene. Svarte streker viser sammenkobling med kabel mens røde streker viser fysisk sammenkobling.

2 Teori og bakgrunnsinformasjon

I skissen over løsningen i figur 1 ser man at det er fire hoveddeler av problemstillingen kamerakalibrering, fargesegmentering, hjørnedeteksjon og robot.

Bildet som blir tatt av kameraet skal kunne gi informasjon gjennom MATLAB og tilbake til roboten som det er festet på. Dette gjør at kamerakalibrering er nødvendig for å kunne oversette punkter i bildet til punkter i virkeligheten og på denne måten finne riktige avstander.

Klossene, eller andre objekter, må kunne skilles fra bakgrunnen. Fargesegmentering er en løsning på dette, samtidig som det gjør det mulig å kunne skille to klosser med ulik farge.

Når en kloss har blitt funnet i et bilde, må egenskapene til klossen finnes for å kunne bestemme hvor og hvordan klossen ligger. Dette blir gjort i hjørnedeteksjonsdelen.

Roboten skal flyttes til punkter i rommet, til dette trengs kommunikasjon mellom MATLAB og Rapid. Kommunikasjonen, generering av arbeidsobjekt og bevegelsene blir gjort i robotdelen.

Det er tatt noen valg som tilsynelatende forvansker programmet ved å skrive en algoritme selv, isteden for å bare bruke eksisterende løsninger. En ting som går igjen i disse valgene er ønsket om å ha større kontroll og forståelse over det som skjer i algoritmene. Det gir også muligheten for å visualisere data samtidig som feilhåndtering blir enklere. Bedre mulighet for å håndtere feil og gjøre endringer i algoritmene gjør de mer robuste.

2.1 Kamerakalibrering

Kalibrering av kamera innebærer å finne de to matrisene K og $[R|t]$ slik at koordinater i bildet kan oversettes til verdenskoordinater og omvendt. Denne oversettelsen bruker øverste hjørne til venstre i bildet som origo for begge koordinatsystemene. Her brukes samme notasjon som brukt i MATLAB. Hentet fra [5].

$$am = M \begin{bmatrix} R \\ t \end{bmatrix} K$$

$$a \begin{bmatrix} u & v & 1 \end{bmatrix} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \\ t_x & t_y & t_z \end{bmatrix} \begin{bmatrix} f_x & 0 & 0 \\ s & f_y & 0 \\ c_x & c_y & 1 \end{bmatrix}$$

Her er a en skaleringsfaktor, u og v er koordinater i bildet og X,Y og Z er koordinater i verden.

K matrisen er kamera matrisen eller intrinsic matrisen. $c_x$og c_y er koordinatene til det optiske senteret i piksler. Parameteren er s, som er skjevheten (skew) mellom aksene, vil være 0 dersom aksene er nøyaktig 90^{o} på hverandre. $f_x$og f_y er fokal-lengden multiplisert med antall piksler i x og y retning.

R matrisen er rotasjonen til kamerakoordinatene i forhold til verdenskoordinatsystemet. Rotasjonsmatrisen er gitt som$R(\alpha, \beta, \gamma) = R_x(\alpha)R_y(\beta)R_z(\gamma)$ der α er rotasjon om X akse, β er rotasjon om Y akse og γ er rotasjon om Z akse.

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix}$$

$$R_y(\beta) = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix}$$

$$R_z(\gamma) = \begin{bmatrix} cos(\gamma) & -sin(\gamma) & 0 \\ sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Dette gir da:

$$R(\alpha, \beta, \gamma) = \begin{bmatrix} c(\beta)c(\gamma) & -c(\beta)s(\gamma) & s(\beta) \\ c(\alpha)s(\gamma) + s(\alpha)s(\beta)c(\gamma) & c(\alpha)c(\gamma) - s(\alpha)s(\beta)s(\gamma) & -s(\alpha)c(\beta) \\ s(\alpha)s(\gamma) - c(\alpha)s(\beta)c(\gamma) & s(\alpha)c(\gamma) + c(\alpha)s(\beta)s(\gamma) & c(\alpha)c(\beta) \end{bmatrix}$$

$$(s() = sin(), c() = cos())$$

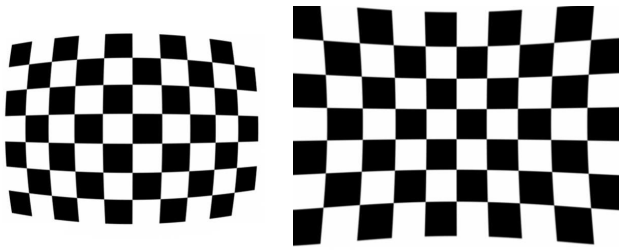
Translasjonsvektoren t er translasjonen eller forflyttingen fra origo i kamerakoordinater til origo i verdenskoordinater. Vektoren er gitt som $t = \begin{bmatrix} t_x & t_y & t_z \end{bmatrix}$, der t_x,t_y og t_z er translasjonen i en enhet (vanligvis mm). Sammen blir R og t vektoren til $\begin{bmatrix} R \\ t \end{bmatrix}$, extrinsic matrisen.

2.1.1 Intrinsic

De indre parametrene beskrevet av K matrisen er ofte lett tilgjengelige gjennom datablad eller EXIF data. Det er likevel tilrådelig å utføre kalibrering av disse parametrene da det ofte er små feil eller ujevnheter i kameraet eller forskjellige typer forvrengning.

I MATLAB 2013b (og nyere) er det kommet et tillegg til Computer Vision toolboxen som heter Geometric Camera Calibration. Denne toolboxen inneholder et grafisk brukergrensesnitt (åpnes med kommandoen *cameraCalibrator*) der bilder av et sjakkrutete mønster legges inn. Ved å vite reell størrelse på sjakkrutene og sammenligne denne med observert størrelse, kan indre og ytre parametre beregnes. Den samme sammenligningen kan brukes for å finne koeffisienter for radial og tangential forvrengning.

Radial forvrengning oppstår i tilfeller der lyset brytes forskjellig avhengig av radius til linsen. Dette gir et bilde med forvrenging avhengig av radius til senter i bildet.



Figur 3: Radial forvrengning. Bildet til venstre viser positiv radial forvrenging (tønneforvrengning) mens bildet til høyre viser negativ radial forvrengning (nålepute).

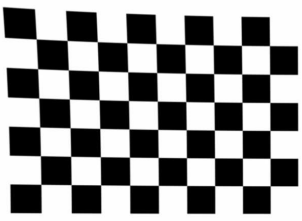
Forvrengingen gjør at piksler i periferien av bildet blir forflyttet nærere eller fjernere fra sentrum. Forholdet mellom egentlig piksel-posisjon, forvrengt piksel-posisjon og forvrengings-koeffisienter kan gis ved:

$$x_{forvrengt} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

$$y_{forvrengt} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

der k_1,k_2 og k_3 er forvrengings-koeffisienter. x og y er egentlig posisjoner mens $x_{forvrengt}$ og $y_{forvrengt}$er forvrengte posisjoner. Radiusen r er gitt som $\sqrt{x^2 + y^2}$.

Tangential forvrenging oppstår når kamerasensoren og kameralinsen ikke er parallelle. Tangential forvrening kan minne om radial forvrenging i deler av bildet siden lyset blir brutt ulikt for disse delene. Årsaken er at linsen og kamerasensoren ikke er vinkelrett på hverandre.



Figur 4: Kombinasjon av radial og tangential forvrengning.

Forholdet mellom piksel-posisjon, forvrengt piksel-posisjon og forvrengningskoeffesienter gis ved:

$$x_{forvrengt} = x + 2p_1 xy + p_2(r^2 + 2x^2)$$

$$x_{forvrengt} = x + p_1(r^2 + 2y^2) + 2p_2 xy$$

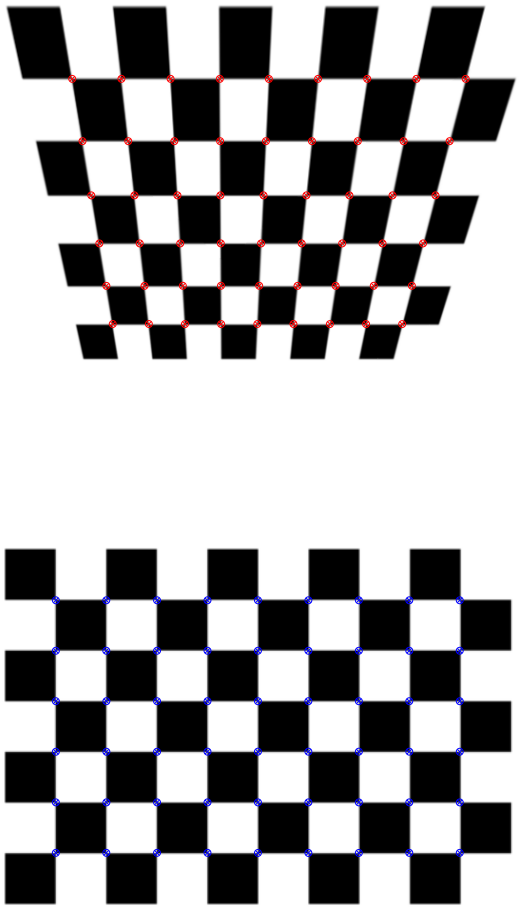
der p_1 og p_2 er forvreningskoeffesienter og r er radius $\sqrt{x^2 + y^2}$. Posisjonene x og y er de egentlige posisjonene mens $x_{forvrengt}$og $y_{forvrengt}$ er de forvrente posisjonene.

Kamerakalibratoren til MATLAB trenger to eller flere bilder av det samme sjakkrutete mønsteret for å kalibrere kameraet. Selv om to bilder er nok, er det anbefalt å bruke mellom 10 og 20 bilder. Et vision.cameraParameters objekt blir da generert ut ifra bildene. Dette objektet inneholder alle forvrenings koeffesienter, intrinsic matrisen samt extrinsic verdier for alle bildene.

2.1.2 Extrinsic

Det er ofte greit å kunne rekalibrere extrinsic parametre uten å ta en full rekalibrering av kameraet. Det vil si at dersom man endrer på omgivelsene til objektene som skal detekteres ved å for eksempel bytte til et annet bord med forskjellig høyde, så kan systemet fremdeles benyttes.

I dokumentasjonen i MATLAB 2013b [6] er prosessen ved å finne extrinsic parametre beskrevet veldig godt med gode eksempler, dessverre er dette fjernet fra MATLAB 2014b der en enkelt funksjon erstatter hele koden. Følgende er delvis hentet fra [6]:



Figur 5: Sjakkmønster og detekterte hjørner. Øverst vises rutemønster med perspektiv for å illustrere et bilde fra virkeligheten og under vises et generert rutenett med samme dimensjoner som over.

To sett med hjørnepunkter for sjakkmønsteret blir sammenlignet, *imageCorners* er punktene fra bildet. *worldCorners* er et generert sjakkmønster som tilsvarer et ideelt mønster av samme dimensjoner som i bildet. Disse punktene kan i teorien også finnes ved å bruke hjørnene på arket som ligger under klossene, men dette er ikke gjort siden man bare får fire punkter å sammenligne med. Funksjonen *fitgeotrans* tar disse to settene med punkter og returnerer en homografien,

en projektiv matrise H.

$$H = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

$$R = \begin{bmatrix} r_1 \\ r_2 \\ r_3 \end{bmatrix}$$

$$\lambda = \frac{1}{||h_1 K^{-1}||} = \frac{1}{||h_2 K^{-1}||}$$

$$r_1 = \lambda h_1 K^{-1}$$

$$r_2 = \lambda h_2 K^{-1}$$

$$r_3 = r_1 \times r_2$$

$$t = \lambda h_3 K^{-1}$$

her er R en 3D rotasjonsmatrise, K er kameramatrisen funnet ved $cameraCalibrator$ og t er translasjonsvektoren.

Rotasjonsmatrisen må tilfredstille visse krav. Determinanten må være 1 og $R^T = R^{-1}$. Grunnet støy kan R være ulik den ekte rotasjonsmatrisen. En tilnærming til den ekte rotasjonsmatrisenR'kan finnes ved singulærverdidekomposisjon.

$$R = U\Sigma V^T$$
$$R' = UV^T$$

Denne korrigeringen er med i funksjonen. T blir så regnet ut som $T = \begin{bmatrix} R' \\ t \end{bmatrix} K^{-1}$ som blir brukt for å generere et $projective2d$ objekt fra $projective2d(T)$ som kan brukes direkte i $transformPointsForward$ og $transformPointsInverse$. Selv om dette gir en komplett transformasjon er den bare oppdatert med gjeldende extrinsic data.

2.2 Fargesegmentering

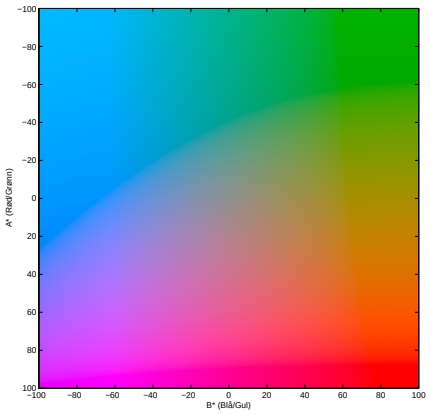
Objekter av med forskjellige farger må kunne skilles fra hverandre samt bakgrunnen. I MATLAB finnes det mange funksjoner for å segmentere objekter på intensitet, for eksempel paret$multithresh(I, n)$ og $imquantize(I, thresh)$ som deler opp et bilde i $n + 1$ segmenter ved å dele opp bildet etter intensitet, som vist i eksempelet fra [9]. En annen metode er å bruke K-means-clustering som vist i eksempelet fra [7]. Metoden som ble valgt bruker en minimum distanse

estimator i L*a*b* fargerommet. L*a*b fargerommet er en tilnærming til måten mennesker ser farger på så farger som mennesker ser som like vil ligge nært hverandre i fargerommet. Minimum distanse estimatoren gir ikke nødvendigvis det beste resultatet, men bruker lite tid. Den er utviklet fra eksempelet funnet på [1].

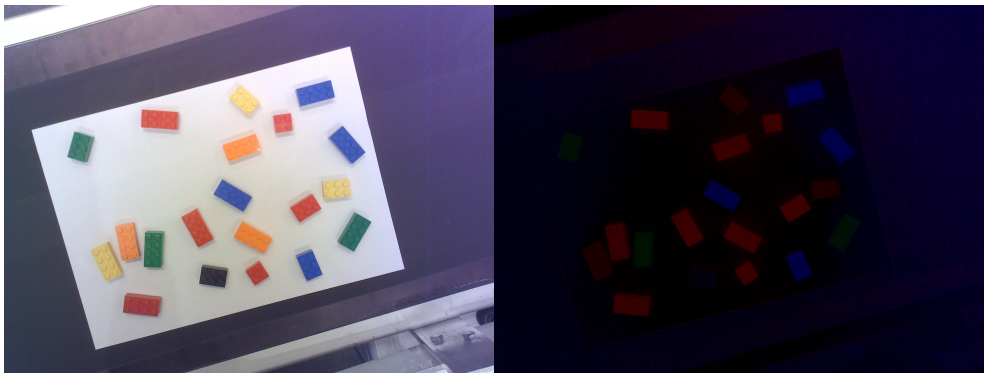
2.2.1 L*a*b* fargerom

L*a*b* fargerommet består av L*, a* og b* komponenter på samme måte som RGB består av rød, grønn og blå. I motsetning til RGB er bare a* og b* kromatiske komponenter, mens L* uttrykker lysstyrke. L*a*b* kan derfor minne om HSV, der v komponenten tilsvarer lysstyrken. Lab og L*a*b* er beslektede fargerom, men bruker forskjellige metoder for farge-transformering [2].

Fargerommet bruker a* og b* for å beskrive farger, a* går da langs rød/grønn aksen mens b* går langs blå/gul aksen som vist i figur6. Dette gjør at alle farger kan beskrives innenfor a*/b* planet uten å ta høyde for intensitet. I figur 7 er L* satt til 0 for å illustrere hvordan fargene tolkes av programmet.



Figur 6: L*a*b* fargerommet med L = 60. Positive a* verdier indikerer rød og negative indikerer grønn, på samme måte gir positive b* verdier gul og negative gir blå.

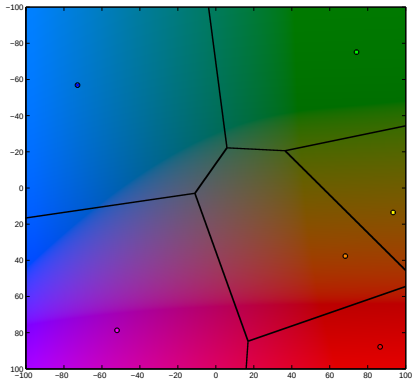


Figur 7: Høyre bilde viser resultat etter å ha transformert fra RGB til L*a*b*, satt L* = 0 og transformert tilbake til RGB. Fargerike områder er lite endret, mens svarte og hvite områder har blitt mørke.

2.2.2 Minimal distanse

Farger kan beskrives med to variabler i a*/b* planet forhold til tre i RGB gamutten. Ved å finne a*/b* gjennomsnitt av områder med kjente farger kan dette gjennomsnittet brukes for å klassifisere ukjente farger. Ukjente farger blir klassifisert ut ifra sin nærhet til kjente farger.

Når et helt bilde skal klassifiseres, brukes denne metoden på hver piksel i bildet. Fargen til pikselen er da den ukjente fargen. Skillet, eller desisjonsgrensen, mellom to farger ligger ortogonalt på midtpunktet mellom to farger. Andre estimatorer gir mer komplekse desisjonsgrenser som kan bety et mer nyansert skille, men bruker lengre tid.

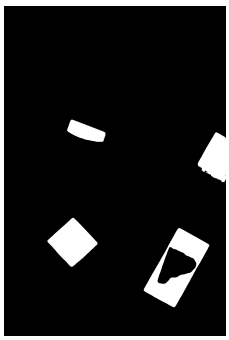


Figur 8: Segmentert fargerom. Seks farger definert som punkter i a*/b* planet. Fargene er merket med farget kryss og sort sirkel. De avgrensede områdene viser klassifisering til nærliggende farger.

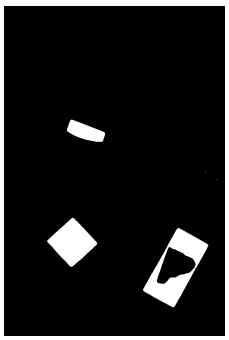
2.2.3 Binærbildebehandling

Etter segmentering er resultatet en rekke binære bilder, en for hver farge. Et binært bilde er matrise der hvert tall bare kan ha to tilstander, 0 eller 1. I MATLAB blir dette visualisert som hvitt eller svart. I de binære bildene finnes det elementer som er uønsket som støy, objekter delvis innenfor bildet og 'hull' i objektene. Ved å fjerne disse blir resultatet en renere klassifisering der usikre objekter blir ignorert.

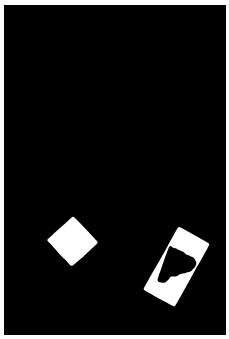
For å fjerne områder i periferien av bildet brukes MATLAB-funksjonen *imclearborder* [13] denne bruker morfologisk rekonstruksjon [16] for å identifisere og fjerne objekter som ligger i kantene. Funksjonen $bwfill$ eller $imfill$ [15] bruker også morfologisk rekonstruksjon, men denne brukes for å fylle hull. Definisjonen på hull er en eller flere 0er fullstendig omgitt av 1ere. Den siste funksjonen er *bwareaopen* [14] som fjerner sammenhengende områder på under et gitt samlet areal. Arealet blir funnet ved *regionprops* mens sammenhengende områder blir funnet ved *bwconncomp*.



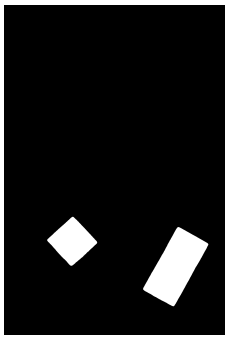
Originalt bilde.



Etter behandling med *imclearborder*.



Etter behandling med *bwareaopen*.



Etter behandling med $bwfill$.

Figur 9: Binærbildebehandling.

2.3 Hjørnedeteksjon

Formen til en kloss kan beskrives ved hjelp av hjørnene, dersom hjørnene er kjent kan også senterpunktet og orienteringen til klossen beskrives. Det samme gjelder for alle rektangulære objekter. Etter fargesegmentering er resultatet et sett med binære bilder med hvite områder der en farge har blitt detektert.

I MATLAB finnes funksjonen *regionprops* [10] som i tillegg til å avgrense området til en enkel kloss blant annet finner ekstremalpunkter, orientering, areal. Den gir da alle de interessante egenskapene som skal brukes videre i prosessering. Denne funksjonen er ikke brukt for å kunne ha større kontroll over hva som skjer, kunne feilsøke og visualisere resultater og for å få større forståelse for de forskjellige algoritmene.

For å kunne detektere hjørnene til hver kloss må man først kunne avgrense området som inneholder en kloss. Dette kan gjøres med *regionprops* som nevnt, eller ved å bruke *bwboundaries* [11].

Effektiviteten til hjørnedeteksjonen kan økes ved bare å se på sidekantene

til klossene. Derfor er det nødvendig med kantdeteksjon før hjørnedeteksjonen. Det finnes svært mange metoder for kantdeteksjon og av disse er *canny* valgt. Canny deteksjon bygger videre på gradientdeteksjon ved hjelp av filterkjerner. Løsningen ble valgt grunnet tidligere kjennskap til algoritmen bak. Et annet alternativ her er å bruke resultatet fra *bwboundaries*, men dette ble valgt bort grunnet formatet på data fra *bwboundaries*.

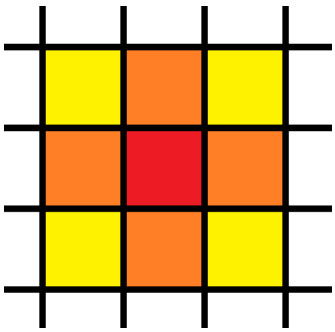
Til den egentlige deteksjonen av hjørnene finnes alternativet *corner*[12] fra MATLAB. Denne funksjonen benytter seg av enten Harris hjørnedeteksjon eller Shi & Tomasi's minimum egenverdi metoden. Alternativet som ble valgt kom fram etter samtale med veileder. Fordelen dette alternativet har fremfor de andre er at resultatet kan visualiseres i en graf som gir muligheten til å 'se' omrisset av objektene fra sentrum av objektet. Denne metoden krever binære bilder og vil ikke fungere på hverken gråskala eller fargebilder.

2.3.1 Boundary trace

Selv om objektene er separert etter farge, eksisterer de bare som boolske verdier i en matrise. For å samle alle piksler som tilhører samme objekt kan man bruke grensesporing (*boundary trace*). MATLAB sin implementasjon*, bwboundaries[11],* bruker Moore-Neighbour tracing algoritmen. Algoritmen fungerer ved å velge en piksel med verdi 1 og rotere rundt denne til man finner en ny piksel med verdi 1. Retningen på denne rotasjonen spiller ingen rolle så lenge den er konsekvent i hele algoritmen. Startpunktet for rotasjonen spiller en stor rolle, man må alltid starte i den forrige gjeldende pikselen, ellers vil ikke algoritmen fungere.

Algoritmen kan bruke nabolag på 4 eller 8 piksler som vist i figur 10. Det er mest hensiktsmessig å bruke 8-piksler i situasjoner der man ønsker en god representasjon av objektet fremfor lav prosesseringstiden.

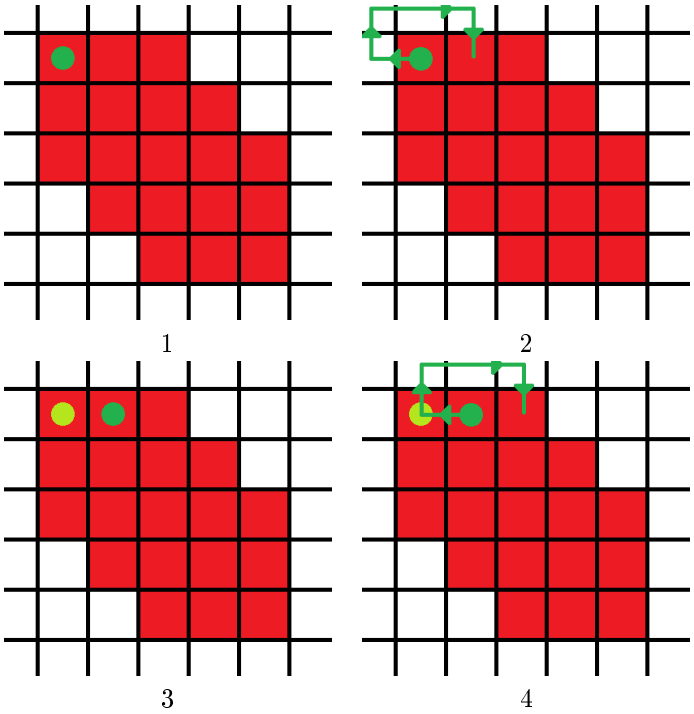
I MATLAB implementasjonen er algoritmen modifisert med *Jacob's stoppkriterie,* som sier at sporingen av objektet er ferdig når man når startposisjonen igjen.



Figur 10: Nabolaget i Moore-Neighbour tracing. Den røde pikselen er startpunktet. De oransje pikslene viser direkte naboer og de gule pikslene viser indirekte naboer. Når nabolaget består av 4 piksler er kun de oransje pikslene inkludert. For 8 piksler er både gule og oransje inkludert.

Algoritme 1 Moore-Neighbour tracing. Hentet fra: [3]

1. Iterer gjennom alle pikslene i bildet til en piksel med verdi 1 blir funnet.
2. Lagre denne pikselen som startpunktet.
3. Lagre startpunktet som gjeldende piksel og legg denne pikselen til i grensepiksler.
4. Gå tilbake til forrige piksel.
5. Roter rundt den gjeldende pikselen til en ny piksel med verdi 1 blir funnet.
6. Lagre denne som gjeldende piksel, legg denne pikselen til i grensepiksler.
7. Gjenta steg 4-6 til man har returnert til startpunktet.



Figur 11: 1: En piksel med verdi 1 blir funnet.(Steg 1-3 i algoritmen) 2: Pekeren går tilbake til den forrige pikselen og roterer rundt pikselen som ble funnet i steg 1 til en ny piksel med verdi 1 blir funnet. (Steg 4-5 i algoritmen) 3: Pekeren flyttes til den nye pikselen. (Steg 7) 4: Gjenta til det første punktet blir funnet igjen.

2.3.2 Kantdeteksjon

For å raskere detektere hjørner blir kantene til objektet funnet. Dette begrenser antall punkter som må behandles betraktelig.

Kantdeteksjon kan gjøres på mange forskjellige måter. Den simpleste metoden er å bruke Sobel eller Prewitt filtre for å finne gradientene til bildet i X og Y retning for så å finne absoluttverdien av retningene.

Canny kantdeteksjon bygger videre på disse teknikkene. Først blir bildet filtrert med et gaussisk filter. Dette er et lavpassfilter som reduser støy i bildet. Deretter finnes gradienten til bildet, vanligvis ved Sobel filtre. som er høypass filtre. Gradient vinklene blir avrundet til 0^o, 45^o, 90^o eller 135^o. For å fjerne piksler som ikke er en del av en kant blir *non-maximum supression* benyttet. Til slutt blir en hysterese med to grenser benyttet. Algoritmen er forklart i detalj i [4]. For å telle som en del av en kant må gradienten til en gitt piksel enten være høyere enn øverste grense eller ha en nabo-piksel som er en del av en kant samt en egen gradient mellom grensene. Dersom en piksel ligger under den nederste grensen blir den avvist.

$$F_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$F_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

$$G_x = F_x * I$$

$$G_y = F_x * I$$

$$G = \sqrt{G_x^2 + G_y^2}$$

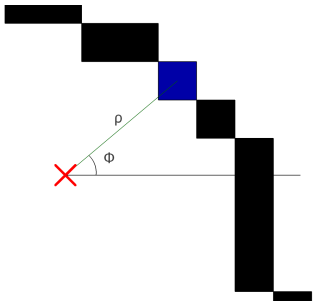
$$\theta = arctan(\frac{G_y}{G_x})$$

Sobel-filtrering der F_x og F_y er Sobel filtrekjerner i henholdsvis x og y retning. G_x og G_y er gradienten til bildet i x og y retning. G er intensiteten eller absoluttverdien av gradienten og θ er vinkelen til gradienten. [17]

2.3.3 Hjørnedeteksjons algoritme

Den valgte hjørnedeteksjons algoritmen baserer seg på å måle distansen fra sentrum til omrisset av objektet. Algoritmen benytter seg av canny kantdeteksjon

for å finne omrisset, samt Moore-Neighbour algoritmen for å segmentere flere objekter i samme bilde. Ved å måle distansen fra sentrum til hver piksel langs kanten av objektet og sortere distansene i forhold til vinkelen ϕ kan objektet uttrykkes som en funksjon av ϕ.

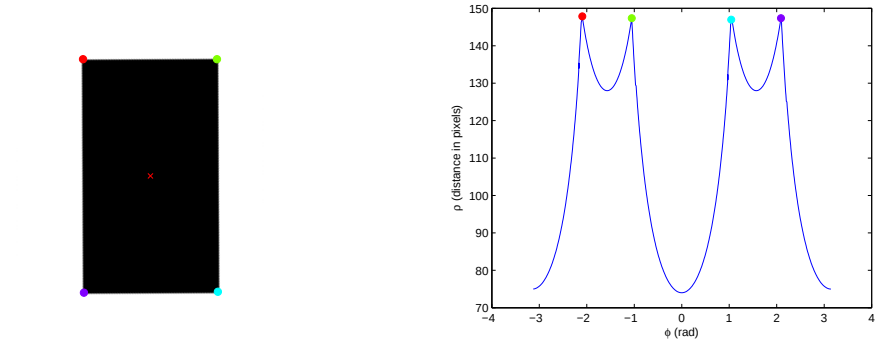


Figur 12: $\rho = f(\phi)$ illustrert. Her er distansen ρ til den aktive pikselen (blå) tegnet inn som en grønn linje. Vinkelen mellom x-aksen og denne linjen er ϕ.

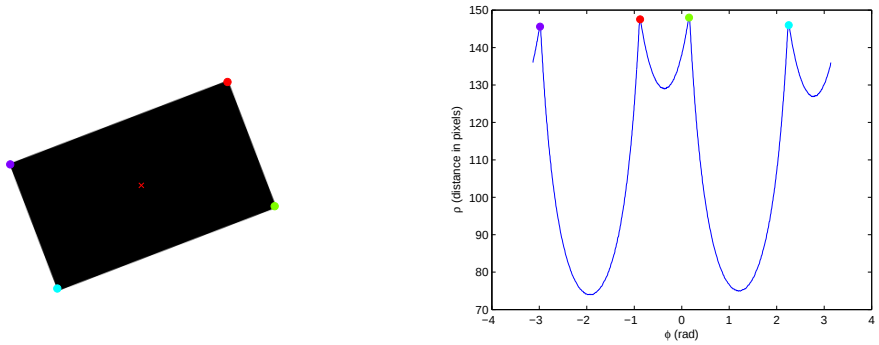
For rektangulære objekter vil hjørner i grafen til $\rho = f(\phi)$ fremstå som topper, siden hjørnene til et rektangel er punktene lengst unna sentrum. Dette kommer tydelig frem i figur 13. Skalering av objektet vil gi endring i amplituden ρ, men ved å normalisere amplituden vil resultatet bli identisk. Ved rotasjon av objektet vil ϕ skiftes sirkulært i samme retning som rotasjonen. Ved å se på vinkelen og amplituden til hver topp kan objekter gjenkjennes bare ut ifra hjørnepunktene. Hjørnene til et rektangulære objekt som ikke er sett direkte ovenfra vil ha ulik distanse til sentrum, dette ble brukt for å verifisere at kameraet ser direkte ned på arket. Dette er vist i figur 14.

Algoritme 2 Hjørnedeteksjonsalgoritmen

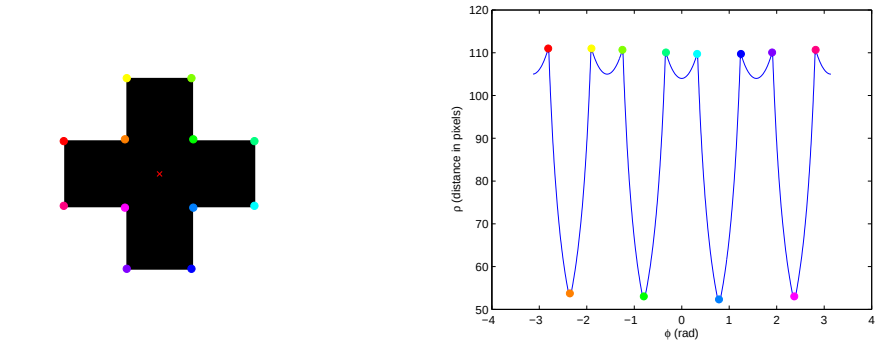
1. Segmenter objektet fra resten av bildet.
2. Detekter kanter og lagre hver piksel som bildekoordinater.
3. Finn sentrum i objektet ved å ta gjennomsnitt av koordinatene til alle kant-pikslene.
4. Subtraher sentrumspunktet fra hver piksel, slik at sentrum i objektet blir origo i et kartesisk koordinatsystem.
5. Konverter de kartesiske pikselkoordinatene polarkoordinater og sorter disse med hensyn på vinkelen θ.
6. Filtrer distansen til senter,ρ, for å unngå kornete signal.
7. Finn toppene til distansen, ρ. Lokale maksima indikerer et hjørne.
8. Dersom fem hjørner er funnet er det mulig objektet er orientert på en måte slik at samme hjørne blir detektert to ganger, skift signalet og prøv steg 7 igjen.
9. Dersom antall hjørner fremdeles er ulikt fire, har objektet for uskarpe kanter eller har mer eller mindre enn fire hjørner.
10. Konverter polarkoordinatene til toppene tilbake til kartesiske koordinater.



Rektangel

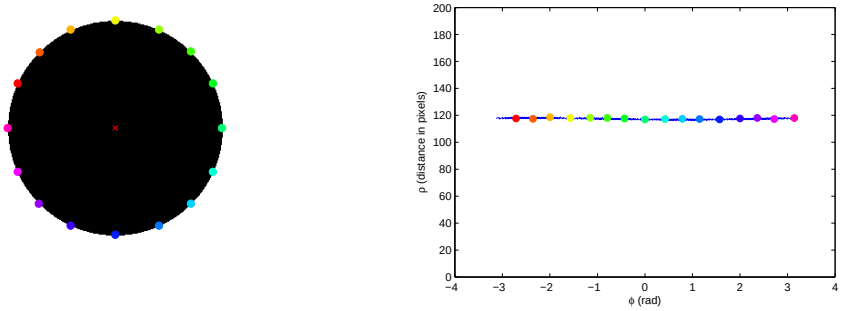


Rotert rektangel

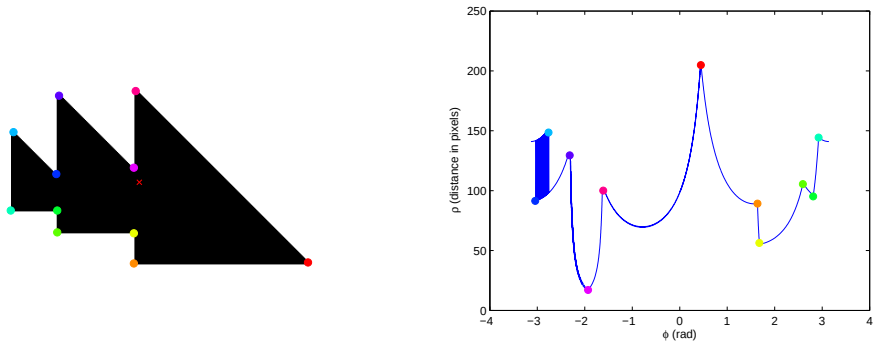


Kryss

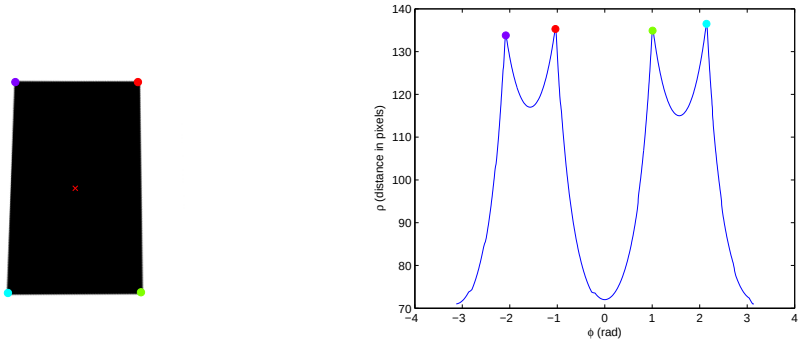
Figur 13: Objekter med klare hjørner og tilhørende $\rho = f(\phi)$ graf. Det er tydelig topp eller bunnpunkt på alle hjørner. Det kommer også fram at rotasjon av objektet tilsvarer et sirkulært skift av ϕ, uten annen endring i grafen.



Sirkel



Sammensatt form



Rektangel med perspektiv.

Figur 14: Algoritmen er ikke like passende for alle former, men kan likevel brukes til gjenkjenning av disse. I bilde-paret med den sammensatte formen kan resultatet av overlappende kanter sees på ca $\phi = -3$. Toppene til rektangelet med perspektiv (sett litt fra siden, ikke direkte ovenfra) har ulik høyde som nevnt i kapittel 2.3.3.

2.4 Robot

Universitetet i Stavanger har en robot-lab med to ABB IRB140 roboter. Den ene av disse robotene, Norbert, er utstyrt med en griper som kan åpnes og lukkes ved hjelp av lufttrykk. Robotene programmeres i programmeringsspråket Rapid gjennom Robot Studio som begge er utviklet av ABB.

Robotene skal kunne plukke opp DUPLO klosser fra en posisjon gitt av MATLAB. For dette trengs kommunikasjon mellom MATLAB og roboten. En potensiell måte å løse dette på er å sende kommandoene til roboten direkte gjennom TCP/IP, men dette krever et enormt arbeid. Derfor ble PC SDK[20], som er en utviklingspakke produsert av ABB, brukt. Pakken er skrevet for .NET og er problematisk å importere inn i MATLAB, den fungerer for eksempel bare med en 32-bits versjon av MATLAB. Et alternativ for å løse noen av problemene er å skrive et ekstra .NET bibliotek som ligger over hver PC SDK som er brukt. Løsningen ble heller å skrive funksjoner i MATLAB som ligger over funksjonene i PC SDK.

Robot kontrolleren er der logikken og rapid programmet ligger. Den er den delen av roboten som styrer bevegelsene.

2.4.1 Rapid

Rapid er et programmeringsspråk utviklet av ABB som består av en rekke instruksjoner og funksjoner som beskriver arbeidet til roboten. Språket skiller mellom funksjoner og instruksjoner i at en funksjon returnerer en verdi, mens en instruksjon bare utfører en oppgave [18].

Til denne oppgaven har er instruksjonene $MoveL$, $MoveJ$, $SetDO$ og $WaitTime$ benyttet [19]. $MoveL$ og $MoveJ$ flytter verktøyet til roboten, begge instruksjonene når samme sluttpunkt, men $MoveL$ vil flytte midtpunktet til verktøyet i en rett linje, mens $MoveJ$ ikke krever at verktøyet beveger seg i ren rett linje. $SetDO$ setter en digital utgang, som brukes til å åpne og lukke griperen. For å vente til griperen faktisk er åpnet eller lukket brukes$WaitTime$ som gir muligheten til å vente i et gitt antall sekunder.

I tillegg til instruksjonene er funksjonene $CRobT$, $OrientZYX$, $RelTool$ og$DefFrame$ benyttet [19].. For å generere et arbeidsobjekt brukes funksjonen $DefFrame$ som krever tre punkter, origo, et punkt på X-aksen og et punkt i XY-planet. Disse punktene finnes ved hjelp av $RelTool$ som gir en posisjon relativ til midtpunktet til verktøyet. For å finne en posisjon relativ til nåværende posisjon må nåværende posisjon være kjent. Denne kan finnes ved $CRobT$ som gir tilbake en *robtarget* variabel. Når midtpunktet av verktøyet er midtpunktet av kameraet, så kan avstanden fra kamerasentrum i verdenskoordinater benyttes direkte i $RelTool$ og slik dannes arbeidsobjektet ut ifra tre punkter i bildet. Rapid bruker quaternioner for å beskrive orientering og for å oversette vinkler til quaternioner brukes $OrientZYX$.

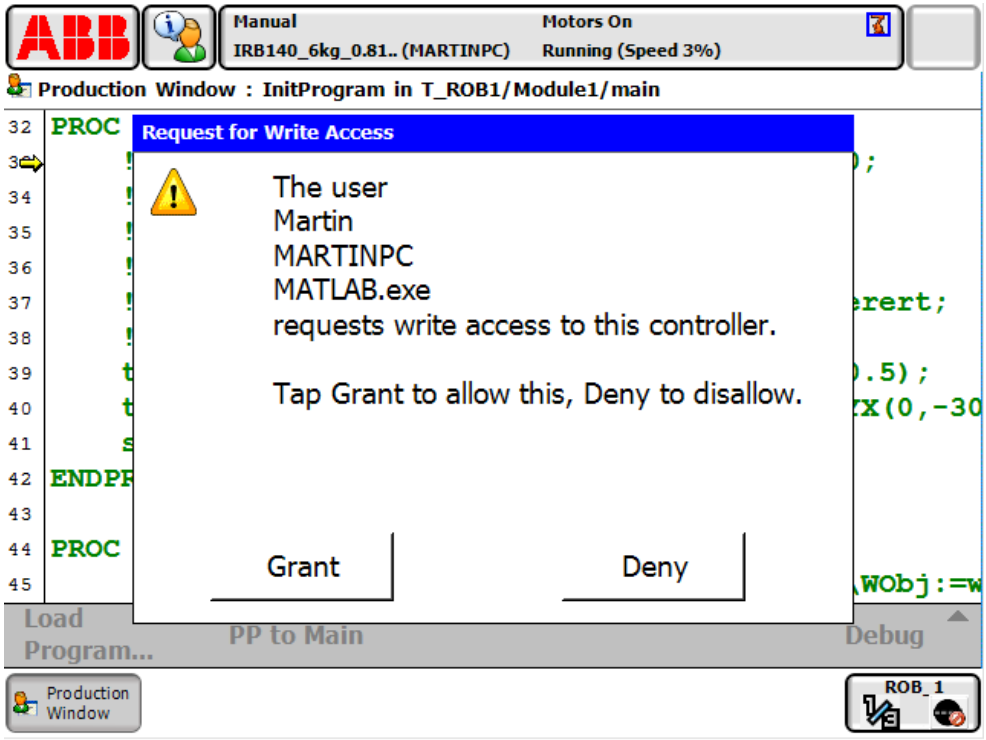
Det er svært mange datatyper i Rapid, de er beskrevet i detalj i[19]. Mange bygger på hverandre og er satt sammen av flere grunnleggende datatyper. I denne oppgaven er num, pos, pose, robtarget, tooldata og workobject benyttet.

num	En numerisk variabel, kan være flyt tall eller heltall. **Eksempel:** -12.75, 154.23 eller 5
pos	En posisjon beskrevet av X, Y og Z koordinater i mm. **Eksempel:** [-12.75,0,154.23] Posisjonen er -12.75 mm langs X-aksen og 154.23 mm langs Z-aksen.
pose	En orientering beskrevet i quaternioner. **Eksempel:** [1,0,0,0] Rotasjonen er 0^{o} om alle tre aksene.
robtarget	Et punkt i rommet. Inneholder en pose som gir posisjonen og orienteringen til midpunktet til verktøyet, samt robconf som gir konfigurasjonen til leddene til roboten og extax som gir posisjonen til eksterne akser. **Eksempel:** [[-12.75,0,154.23],[1,0,0,0],[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]] Posisjonen er -12.75 mm langs X-aksen og 154.23 mm langs Z-aksen. Det er ingen rotasjon. Konfigureringen sier at den fjerde aksen skal være i -1 kvadranten. De eksterne aksene er ignorert siden de er satt til 9e9
tooldata	Beskriver midpunktet, vekten, massesenter, treghetsmoment til et verktøy samt om verktøyet er holdt av roboten. **Eksempel:** [TRUE,[[-12.75,0,154.23],[1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]]; Dette verktøyet er holdt av roboten. Det har midtpunktet 154.23 mm rett ut fra monteringspunktet til verktøyet og -12.75 mm i X akse. Verktøyet har rotasjonen [1,0,0,0], som betyr at det ikke er rotert i forhold til monteringspunktet. Vekten til verktøyet er 1 kilo, massesenteret er 1 mm ut fra monteringspunktet og massesenteret er ikke rotert i forhold til monteringspunktet. Verktøyet kan sees på som en punktmasse uten treghetsmoment.
workobj	Et arbeidsobjekt. Beskriver et koordinatsystem i forhold til arbeidsobjektet wobj0, som tar utgangspunkt i monteringspunktet til roboten. **Eksempel:** [FALSE, TRUE, "", [[-12.75,0,154.23],[1,0,0,0]],[[0, 0, 0],[1, 0, 0, 0]]]; Roboten holder ikke arbeidsobjektet. Koordinatsystemet er låst, altså ikke bevegelig. Ingen mekaniske enheter koordinerer roboten sine bevegelser. Posisjonen til arbeidsobjektet er -12.75 mm i X akse og 154.23 mm i Z akse og er ikke rotert i forhold til wobj0. Objektkoordinatene er de samme som brukerkoordinatene.

Tabell 1: Forskjellige datatyper brukt i programmet

2.4.2 PC SDK

Kommunikasjonen mellom Rapid og MATLAB bruker ABB sin PC SDK pakke. Pakken inneholder en rekke biblioteker eller assemblies skrevet i C# som kan brukes av MATLAB. Når bibliotekene har blitt lastet inn i MATLAB finnes de i minnet og kan ikke slettes igjen uten å restarte MATLAB, noe som kan være problematisk. Til oppgaven er det skrevet en rekke funksjoner som ligger som et ekstra lag over PC SDK assemblyene i MATLAB for å gjøre kommunikasjonen enklere.

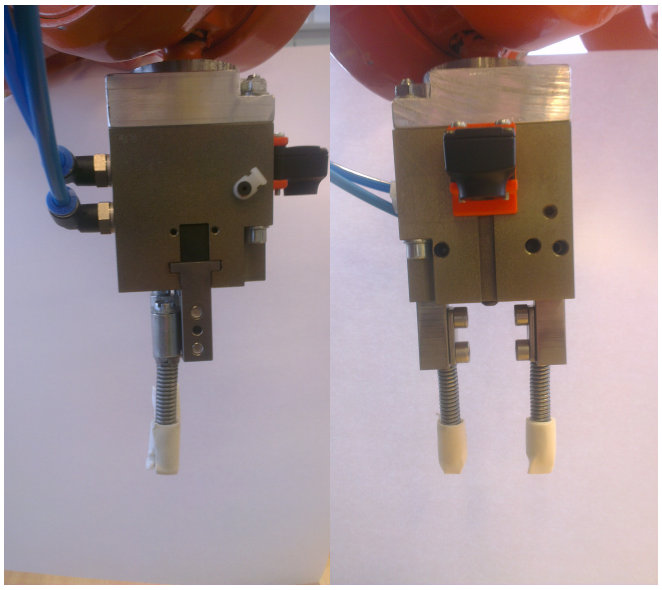


Figur 15: Forespørsel om skrivetilgang.

PC SDK gir muligheten til å dele en variabel mellom MATLAB og Rapid. For å kunne skrive til en variabel i Rapid, må man først logge inn som master. Når MATLAB spør om master tilgang kommer en forespørsel opp i vinduet på FlexPendanten, som er det menneskelige brukergrensesnittet til roboten. Ved å godta forespørselen blir MATLAB satt som master og får skrivetilgang til alle variablene i Rapid programmet.

2.4.3 Gripeverktøy

Roboten med navnet Norbert har en griper som åpnes og lukkes med lufttrykk. Griperen har et trapesformet spor som gjør at andre verktøy enkelt kan monteres. Kameraet er montert på et 3D-printet festeanordning som festes på griperen i sporet. Det er dyttet helt opp i sporet slik at det alltid er i samme posisjon.



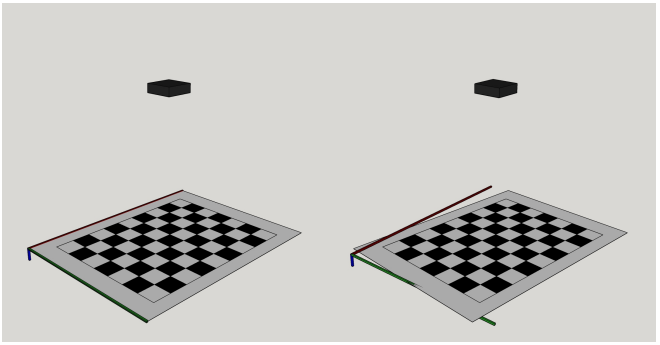
Figur 16: Griperen til roboten med gripefingre og kamera.

Mellomrommet mellom griperen er for stort til å lukkes om en DUPLO kloss. I tillegg vil griperen lage merker i klossene. Etter samtale med Ståle Freyer ble løsningen konstruert med fjærede gripefingre som monteres rett på griperen.

2.4.4 Tooldata

Rapid programmet må vite hvor midtpunktet av et verktøy ligger, dette samt rotasjonen i forhold til monteringspunktet lagres i en tooldata variabel. For griperen ligger midtpunktet midt mellom gripefingrene, for kameraet ligger midtpunktet i sentrum av kameralinsen.

For å generere et arbeidsobjekt vist i figur 17 brukes avstander relativt til kameraet . Figuren viser at selv om kameraet har riktig intrinsic og extrinsic parametre, vil alt bli feil dersom sensoren ikke ligger vinkelrett på kamerahuset.



Figur 17: Arbeidsobjekt med korrekt og unøyaktig tooldata.

For å unngå denne feilen må tooldataen i Rapid programmet reflektere rotasjonen og translasjonen til kamerasensoren i forhold til monteringspunktet. En tooldata variabel er definert som

$$tooldata = [robhold, tframe, tload],$$

der tframe viser rotasjon og translasjon i forhold til verktøyets monteringspunkt, definert i Rapid som tool0.

Variabelen tframe er av typen pose og er definert som

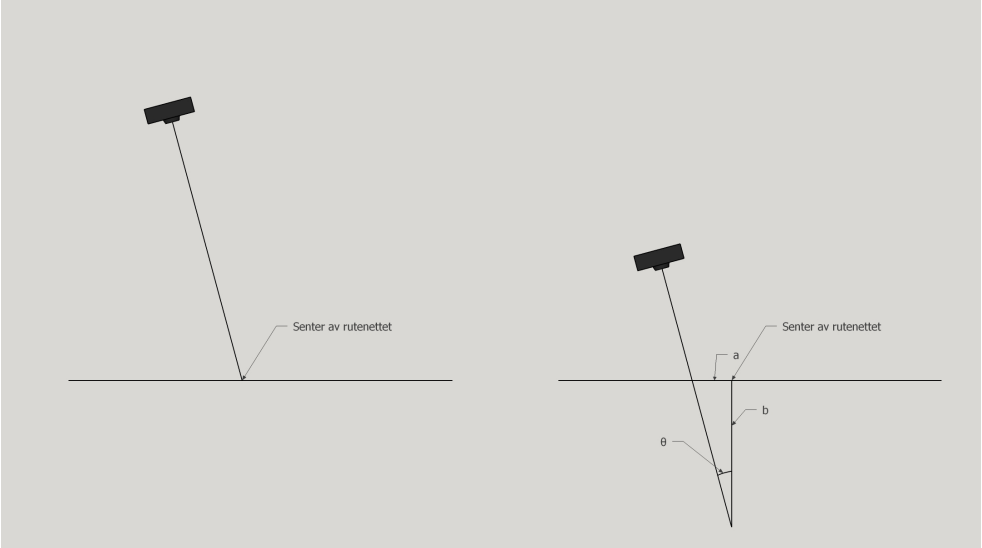
$$tframe = [[X, Y, Z], [Q1, Q2, Q3, Q4]]$$

der X,Y og Z er forskyvelsen fra senter av monteringspunktet til sentrum av verktøyet og Q1,Q2,Q3 og Q4 uttrykker rotasjonen til senteret av verktøyet i quaternioner. For å finne forskyvelsen er databladet til kameraet og skyvelær brukt. Rotasjonen krever noe mer arbeid.

Rotasjonen til verktøyet blir da den indre rotasjonen til kamerasensoren. Den kan finnes ved å benytte et rutenett og bevege kameraet i diverse retninger. Rotasjon om X og Y aksen finnes ved å bevege kameraet opp og ned og måle forflyttelsen i X og Y akse. Dette må kanskje gjøres en gang for hver akse, avhengig av størrelseorden på rotasjonene. Ved invers tangens finnes vinkelen $\theta = atan(\frac{a}{b})$ for begge aksene, der a er forflyttingen til kamerasenteret langs aksene og b er forflyttingen langs Z-aksen. Deretter regnes rotasjonen om til quaternioner og skrives inn i tooldataen.

Rundt Z-aksen er rotasjonen noe mer innfløkt. Her må aksene til arbeidsobjektet sammenlignes med aksene i virkeligheten. Først og fremst må rotasjonen rundt både X og Y akse være tilnærmet null. Deretter sammenlignes aksene i et generert arbeidsobjekt med aksene i virkeligheten. For å generere et arbeidsobjekt kan sidene på et ark med rutenett benyttes, med denne metoden kan rutenettet benyttes direkte. Kameraet beveges da langs arbeidsobjektets X-, eller Y-akse, med utgangspunkt i et kryss på rutenettet. Dersom rotasjonen rundt Z-aksen er ulik null, vil kamerasenteret forflyttes langs motsatt akse betydelig. Vinkelen finnes på samme på måte som for X og Y-aksen, ved $\theta = atan(\frac{a}{b})$ der

b er forflyttingen langs den valgte aksen (det store tallet) og a er den uønskede forflyttingen langs motsatt akse (det lille tallet).



Figur 18: Kalibrering av tooldata for X og Y akse.

3 Implementering

For å detektere og plukke opp DUPLO klosser blir programmeringsspråkene MATLAB og Rapid benyttet. Rapid er et programmeringsspråk laget spesifikt til ABB roboter. Til kommunikasjon mellom MATLAB og Rapid brukes ABB PC SDK, som er en samling av .NET biblioteker som kan benyttes i MATLAB. Roboten, som er av type IRB-140, bruker en griper med fjærbelastede gripefingre for å plukke opp DUPLOen og et kamera for å se DUPLOen.

Prosessen begynner med kameraet. Dersom det eksisterer kalibrering for intrinsic og extrinsic parametre til kameraet blir det tatt et bilde med bildeplanet parallelt med verdenskoordinatene. Dersom det blir detektert et papirark i bilde brukes hjørnene til arket for å lage et arbeidsobjekt i Rapid. Dette arbeidsobjektet er bindeleddet mellom virkeligheten, roboten og bildet fra kameraet og tar utgangspunkt i at øverste hjørne til venstre av arket (i bildet) er origo for koordinatsystemene i både MATLAB og Rapid. Roboten flytter seg deretter til sentrum av arket, slik at arket er vinkelrett i bildet, og tar et nytt bilde. DUPLO klossene i det nye bildet blir klassifisert ved hjelp av fargesegmentering og hjørnepunkt, sentrum og vinkel i forhold til arket blir detektert og lagre i en struktur. Deretter flyttes roboten til alle senterpunktene, roteres med vinkelen og flyttes ned til klossen før griperen lukkes og klossen plukkes opp.

For å løse problemstillingen har en rekke funksjoner blitt skrevet i MATLAB. Disse funksjonene er skrevet på en generell måte, slik at de kan benyttes i fremtidige prosjekter som involverer fargesegmentering, hjørnedeteksjon og kommunikasjon med ABB robot. For kjente feilkilder er det definert beskrivende feilmeldinger, samt håndtering av vagt definerte parametre. For eksempel blir $abb.connect('localhost')$ tolket som $abb.connect('127.0.0.1')$, da dette er en vanlig norm innenfor nettverks-kommunikasjon. Mange av funksjonene har også funksjonalitet utover det som er benyttet i oppgaven, nettopp for å kunne benyttes i fremtidig arbeid. For eksempel kan alle resultater fra $abb.get()$ returneres både som en vektor som tilsvarer data i Rapid, eller som en struktur der de forskjellige parametrene blir beskrevet på en mer menneskelig forståelig måte.

Alle funksjoner er navngitt i camelcase der første bokstav er liten, mens hvert nye ord har stor bokstav. Funksjoner som brukes direkte i kommunikasjon med ABB robot bruker en pakke, de har alle prefiksen $abb.<funksjonsnavn>$, for lettere å skille de fra resten av funksjonene. Dette oppnås ved å legge funksjonene i en mappe med navnet $+abb$ i rotmappen til programmet. Dersom rotmappen er i MATLAB stien blir funksjonene alltid tilgjengelige med $abb.*$.

3.1 Kamerakalibrering

Kamerakalibreringen er sentral for at programmet skal fungere, uten denne er alt fra kameraet unøyaktig eller feil. Den blir utført ved hjelp av *cameraCalibrator* funksjonen i MATLAB 2014, som gir tilbake en struktur som inneholder all nødvendig data. Det er bare nødvendig å kalibrere en gang for hvert kamera, så lenge kalibreringsdata er tilgjengelig er det unødvendig å gjøre det på nytt. Det trengs også en kalibrering av extrinsic data, som gir informasjon om distanse og

rotasjon fra kameraet til ønsket koordinatsystem, denne kalibreringen må gjøres hver gang en ny 'scene' blir tatt i bruk. Kalibrering fra hvert av disse ligger som i filen *calib\calibration.mat* og lastes inn når programmet starter.

Det vil være nødvendig å kalibrere extrinsic parametre på nytt dersom et annet bord blir benyttet siden dette endrer den kjente høyden. For å kalibrere extrinsic parametre på nytt brukes funksjonen $getTransform$ og et bilde av et sjakkrutemønster i riktig høyde. For rotasjon rundt Z-aksen og forskyvning langs X og Y aksen er det ikke nødvendig å rekalibrere siden dette blir korrigert når arbeidsobjektet blir generert. All bevegelse til roboten er avhengig av det genererte arbeidsobjektet og dette er da korrigert. For å finne en distanse mellom to punkter i bildet brukes funksjonen $getWorldDistance$ som gir ut distanse i X og Y retning i verdenskoordinater.

3.1.1 getFrame

Denne funksjonen returnerer ett enkelt bilde (frame) fra et valgfritt tilkoblet kamera som tilhører *winvideo* adapteret. Andre adaptere inkluderer *kinect* og *matrox* dersom drivere for disse er installert. Enheter som kan brukes som webcamera, interne eller eksterne, ligger innenfor *winvideo* adapteret (for mac heter dette adapteret *macvideo*) dersom drivere er installert.

Bildet som blir returnert er i standard format fra adapteret, dette formatet kan være i varierende oppløsning og fargerom avhengig av adapteret. For RGB fargerommet er returverdien en $UxVx3$ matrise, der U og V er antall piksler langs hver akse, og tallet 3 angir antall fargekanaler. For alle testede adaptere og formater er fargekanalene 8 bit, dette gir en 24 bits fargeoppløsning noe som tilsvarer 16777216 forskjellige farger. Før bildet blir returnert venter funksjonen i 3 sekunder for å gi kameraet tid til å initialisere seg, noe som er nødvendig for det brukte kameraet.

Funksjonen har mulighet for å vise en forhåndsvisning av data direkte fra kameraet, før et bilde blir returnert. Denne forhåndsvisningen har et kryss i sentrum av bildet, fire kryss litt utenfor sentrum og fire kryss i periferien av bildet. Kryssene litt utenfor sentrum er forskjøvet 100 piksler i U og V retning. Kryssene i periferien av bildet er i sentrum for ene aksen og maks/min for andre aksen, slik at de danner et stort kryss som går gjennom sentrum av bildet. Krysset i sentrum er der for å lettere ha et utgangspunkt for målinger, mens kryssene i periferien er for å lettere detektere feil. Dersom rotasjonen om Z aksen er feil er det mye lettere å se dette dersom punkt langs som spenner over hele bildet. Kryssene rett utenfor sentrum danner et kvadrat og brukes å kunne sammenligne kvadrater i virkeligheten med et kvadrat i bildet. Dersom de ikke er kvadratiske i bildet er kameraet rotert i X eller Y retning. Alle kryssene gir dermed en rask og enkel tilbakemelding om eventuell feiljustering av kameraet i forhold til det observerte objektet, noe som er nyttig i feilsøkning. Bildet blir returnert når en knapp blir trykket ned.

Funksjonen har to parametre, $winvideoID$ og $enablePreview$. Tallet $winvideoID$ brukes for å velge kamera, dersom det er installert mer enn ett kamera. Forhåndsvisningen er aktivert dersom $enablePreview$ eksisterer og er ulik 0.

For å bruke det første installerte kameraet uten forhåndsvisning brukes funksjonen slik $getFrame(1)$, mens for å bruke funksjonen med forhåndsvisning brukes $getFrame(1,1)$.

3.1.2 getTransform

MATLAB sin Camera Calibration Toolbox har funksjonene $transformPointsInverse$ og $transformPointsForward$. Disse funksjonene trenger et transformasjons-objekt for å regne ut punkt fra ett koordinatsystem til et annet. Det finnes en rekke forskjellige transformasjons-objekter, for eksempel affine2d, affine3d og projective2d. Mye av koden i denne funksjonen, samt dokumentasjon, er hentet fra et eksempel i MATLAB 2013b dokumentasjonen [6], i MATLAB 2014b er den interessante koden i eksempelet erstattet av en enkel funksjon $[R,t] = extrinsics(imagePoints, worldPoints, cameraParams)$.

Funksjonen $getTransform$ har noe likt bruksområde som $extrinsics$, men $extrinsics$ returnerer $[R,t]$ der $getTransform$ returnerer translasjonsmatrisen funnet ved $tform = projective2d([R(1,:), R(2,:); t] * inv(K))$;

Funksjonen har tre parametre, I,$squareSize$ og $cameraParameters$ der I er bildet med sjakkmønsteret, $squareSize$ er størrelsen på rutene og $cameraParameters$ er et $vision.cameraParameters$ objekt som inneholder kalibreringsparametre for kameraet. Returverdien er et $projective2d$ objekt. Algoritmen er beskrevet i større detalj i kapittel 2.1.2.

3.1.3 getWorldDist

Rapid programmet bruker *reltool* funksjonen for å orientere kameraet i forhold til bildet som er tatt. *Reltool* bruker en offset i forhold til verktøyet (kameraet) i X,Y og Z retning. For å finne distansene i verdenskoordinater mellom to punkt i bildet brukes $getWorldDist$, som returnerer distansene i X og Y retning.

Distansene i verdenskoordinater, dx og dy finnes ved å dekomponere vektoren $\vec{AB}$, der A og B er start og stoppunkt. Punktene x og y som vist i figur 19 blir så brukt til å finne X_{verden}og Y_{verden}, ved hjelp av $transformPointsInverse$ funksjonen. Disse punktene sammen med A_{verden}brukes for å finne distansene

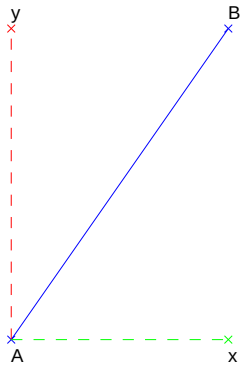
$$dx = \sqrt{A^2_{verden} - X^2_{verden}}$$

og

$$dy = \sqrt{A^2_{verden} - Y^2_{verden}}$$

. Det korrekte fortegnet til distansen finnes ved å bruke $sign()$ funksjonen til MATLAB.

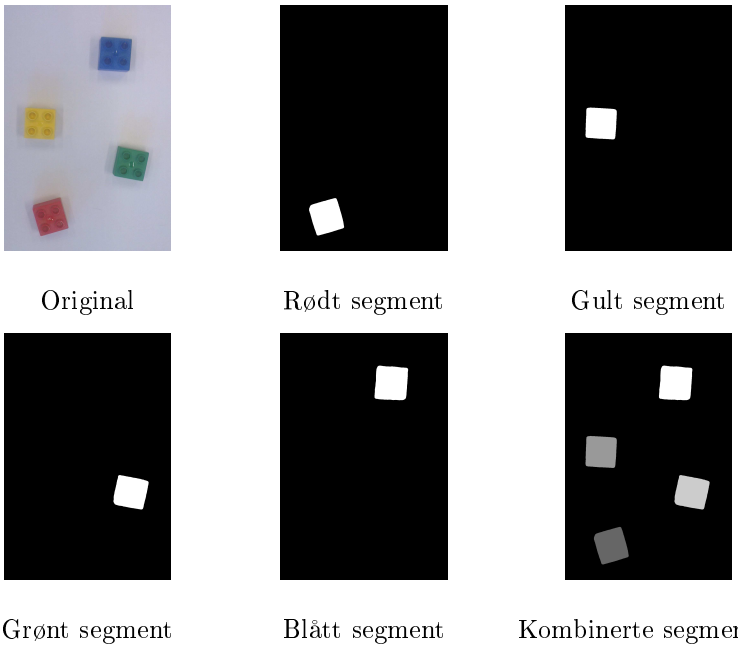
Funksjonen returnerer offsettene dx og dy i verdenskoordinater som kan brukes direkte i Rapid med *reltool*. Inngangsparametrene er *pictureStart*,*pictureStop* og $tform$, der *pictureStart* og *pictureStop* er punktene i bildet der distansen skal finnes og $tform$ er et $projective2d$ objekt fra $getTransform$ funksjonen.



Figur 19: Punktene brukt av getWorldDist.

3.2 Fargesegmentering

For å se forskjell på de forskjellige DUPLO klossene blir bildet fra kameraet delt opp i forskjellige binære bilder avhengig av fargen i bildet. Fargesegmenteringen bruker distansen i L*A*B* fargerommet fra kjente farger til ukjente farger for å klassifisere fargen. L*A*B* fargerommet er laget for å tilsvare det menneskelige øyet. Gjennom denne sammenligningen blir en farge er nærmest den kjente fargen gul blir fargen klassifisert som gul. Klassifiseringen gir områder i fargerommet som er klassifisert som kjente farger, der en linje blir dratt midt mellom to kjente farger. De kjente fargene er gjennomsnittet av et valgt område funnet med *getColorMarkers* og klassifiseringen blir utført av *getSegmentedImage*. Etter klassifisering returnerer *getSegmentedImage* en UxVxN matrise der U og V er oppløsningen til det originale bildet og N er antall farger. Hver farge får sitt eget binære bilde, i tillegg til dette returnerer funksjonen en UxV matrise der verdien av hver piksel er klassifiseringen.



Figur 20: Alle resultater fra fargesegmentering. Sidekanten til den røde klossen er detektert som en topp, noe som gir noe unøyaktighet.

3.2.1 getColorMarkers

For å bruke *getSegmentedImage* til å segmentere objekter i bildet trengs definisjoner av fargene. Disse kan hentes ut ifra bildet direkte ved å markere områder med korrekt farge gjennom funksjonen *getColorMarkers*. Funksjonen bruker *roipoly* for å definere polygoner som inneholder riktig farge, og finner gjennomsnittet av a* og b* komponentene innenfor området.

Funksjonen har tre parametre, I, $nColors$, og $names$. Der I er bildet fargene finnes i, $nColors$ er antall farger og $names$ er en cell array med navnene på fargene. Dersom $names$ er definert vil navnene til fargene vises når polygonene skal velges. Returverdien er en matrise med dimensjonene $2 \, x \, nColors$ som inneholder a* og b* komponentene til hver farge.

3.2.2 getSegmentedImage

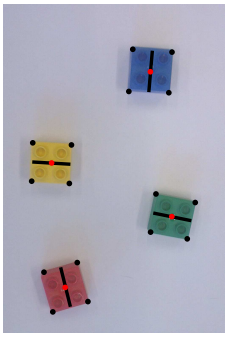
Segmenteringen bruker en klassifiseringsalgoritme for å klassifisere hver piksel av bildet i forhold til en rekke kjente farger hentet fra *getColorMarkers*. I *getSegmentedImage* blir distansen fra de kjente fargene til hver piksel i L*a*b* fargerommet funnet. Den minste distansen gir klassifiseringen av pikselen. Bildet blir filtrert med filterkjernen h, før klassifisering. Dersom h ikke blir gitt gjennom parametrene blir det satt til en 25x25 gaussisk filterkjerne med $\sigma = 5$, dette er en ganske grov filtrering og er gjort for å blande ut fargene litt før gjennomsnitting.

Klassifiseringen gir tilbake like mange binære bilder som det fins kjente farger. Bildene blir behandlet med *imclearborder* for å fjerne objekter fra kantene av bildet, *bwareaopen* for å fjerne mindre objekter og $bwfill$ for å fylle hull i objektene. Objekter i kantene blir fjernet for å unngå delvis klassifisering der halvparten av et objekt er utenfor bildet. Områder med et areal på mindre enn totalt antall piksler delt på 3000blir fjernet, for områder under dette er mest sannsynlig støy eller misklassifiserte omriss. Det er ofte hull innenfor objekter fra refleksjoner eller 'knotter' på DUPLO-klossene, så lenge hullene har et omriss vil $bwfill$ fylle hullene.

Funksjonen har tre inngangsparametre, I, $colorMarkers$ og h. Der I er bildet som skal klassifiseres, $colorMarkers$ er fargene fra $getColorMarkers$ og h er en egendefinert filterkjerne. Returverdiene er $images$ og $combinedImages$. Matrisen$images$ er en binær matrise med størrelsen UxVxN der U og V er oppløsningen til det originale bildet og N er antall farger. For å se det første segmentet brukes $images(:,:,1)$. Matrisen $combinedImages$ er en kombinasjon av alle $images$ der $images(:,:,1)$ har verdien 1, $images(:,:,2)$ har verdien 2 og så videre opp til $images(:,:,n)$ som har verdien n. Denne matrisen kan brukes for å feilsøke eller visualisere klassifiseringen og kan vises med $imagesc$ eller $imshow(combinedImages, [])$ som begge skalerer verdiene i bildet.

3.3 Hjørnedeteksjon

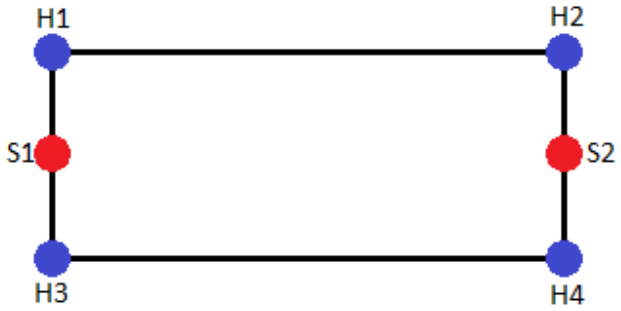
Hjørnedeteksjon er brukt for å finne hjørnene av klosser og papirarket klossene ligger på. Deteksjonen fungerer ved å finne distansen fra sentrum av et objekt til pikslene langs sidene av objektet og pikslene med høyest distanse blir detektert som hjørner. Til dette blir $getCorners$, $getCornersPaper$ og $getCornersFromImages$ brukt. Forskjellen i funksjonene ligger i at $getCornersPaper$ er laget for å finne hjørner på et papirark og innholdet på arket og $getCornersFromImages$ er laget for å finne hjørner i sett av binære UxVxN matriser fra $getSegmentedImage$.



Figur 21: Plottede hjørner, sentrer og vinkler fra $getCornersFromImages$.

For å finne relativ posisjon til hjørnene (topp venstre, topp høyre, bunn

venstre, bunn høyre vist i figur 22) brukes distansen fra origo av bildet (øverst til venstre). Det øverste hjørnet til venstre av kossen, H1, blir funnet ved å finne hjørnet med kortest distanse til origo. H4 blir funnet ved å finne hjørnet med størst avstand til H1. H2 har nest størst avstand til H1 og H3 har minst avstand til H1. Punktene S1 og S2 blir funnet ved gjennomsnitt av X og Y komponentene til H1+H3 og H2+H4. Dersom en kloss blir detektert med sidekanter (som den røde klossen i figur 20) vil feilen bli korrigert ved at sidekanten blir en del av den lengste siden som vist i figur 21. Når roboten plukker opp klossen vil den alltid gripe over den korteste siden.



Figur 22: Posisjonering av hjørnene til rektangelet.

3.3.1 getCorners

Mye av programmet ligger i å finne hjørner i binære bilder. For dette er funksjonen getCorners skrevet. Algoritmen til funksjonen er beskrevet i kapittel 2.3.3. Funksjonen er skrevet for rektangulære objekter, men returnerer hjørnepunkt og sentrum uansett hvor mange hjørner som blir funnet. Dersom antall hjørner er ulikt fire vises graf over $\rho(\phi)$, dette feilsøking lettere og mer visuelt.

Funksjonen har ett parameter men returnerer tre verdier. Parameteren er det binære bildet der hjørner skal finnes, dette bilde må bare inneholde ett objekt, så det må allerede være segmentert. Returverdiene er X- og Ykoordinater for hjørnene samt c som er koordinater for senteret av objektet. Koordinatene X og Y er i forhold til bildet brukt som inngangsparameter. For hovedbilde med flere objekter betyr dette at koordinatene X og Y må summeres med koordinatene til segmentet for å få globale bildekoordinater.

For å unngå falske positiver er det satt to krav til hjørnene. Distansen til et hjørne må være minimum 90% av maks distanse og det må være 45^{o} mellom hvert hjørnepunkt. Det betyr at funksjonen ikke vil detektere svært avlange rektangler.

3.3.2 getCornersPaper

Programmet bruker et papirark for å definere arbeidsobjektet. For å finne papirarket må alle fire hjørnene av arket være innenfor bildet samt at ingenting kan ligge på kanten av arket. Arket må være tilnærmet rektangulært med klare hjørner og sidekanter, men det er en liten margin for feil. Funksjonen *getCornersPaper* finner arket og returnerer koordinater og sentrum på samme måte som *getCorners* samt innholdet av papirarket med bakgrunnen klippet bort.

For å finne arket blir bildet I gjort om til et gråtone bilde og alle piksler med en verdi under en satt terskel blir satt til 0, mens alt over blir satt til 1. På denne måten genereres et binært bilde som tilsvarer omrisset av arket. Objekter på arket kan lage områder under terskelen og for slike hull blir de tettet med $bwfill(bilde,'holes')$ funksjonen. Det kan være områder i bildet som ikke er en del av arket men som likevel kommer med etter tersklingen, disse blir fjernet med $bwareaopen(bilde, maxSize)$, der $maxSize$ er satt til 10% av størrelsen av bildet. Dette betyr at arket må være større enn 10% av bildet for å bli detektert.

Inngangsparameteren er bildet I som blir behandlet, utgangsparametrene er X,Y og c, der X og Y er sett med koordinater til hjørnene og c er sentrum av arket.

3.3.3 getCornersFromImages

For å hente ut hjørnene av objekter i en binær bildematrise fra *getSegmentedImage*, brukes *getCornersFromImages*. Funksjonen returnerer en struktur inndelt i farger og objekter, der fargene er i samme rekkefølge som bildene ligger i. Objektene i de binære bildene blir hentet ut ved å bruke *bwboundaries* og bruke maksimal og minimal piksel-verdi i horisontal og vertikal retning. På den måten kan et bilde av bare objektet hentes ut. Det er lagt til en ramme på 5 piksler rundt objektet for å unngå at objektet ligger helt ute i kanten av bildet. Deretter brukes *getCorners* for å finne hjørnene av objektene. For sikkerhets skyld blir ikke objektet lagt til i strukturen med mindre det har fire hjørner.

Funksjonen har to inngangsparametre, *images* og *names*, der *images* er i samme format som returverdien fra *getSegmentedImage* og *names* er en cell array med navn på fargene. Returverdien er en struktur med to kolonner, name og object. I object ligger en ny struktur med alle hjørnepunktene, vinkler og senterpunktene til alle objekter under hver farge. For å finne hjørnet av det første røde objektet fra eksemplene i tabellene 2 og 3, brukes $col(2).object(1).corners$. Denne strukturen er lagd for å lettere kunne iterere over alle objekter og for å kunne behandle objektene avhengig av farge og posisjon.

name	object
'Background'	[]
'Red'	1x3 struct
'Blue'	1x2 struct

Tabell 2: Eksempel på returstrukturen fra getCornerFromImages.

corners	center	angle
[1 1; 4 4; 1 4; 4 1]	[2.5 2.5]	0
[5 5;7 7; 5 7; 7 5]	[6 6]	0

Tabell 3: Eksempel på objektstrukturen i returstrukturen.

3.4 Robot

Kommunikasjonen bruker biblioteker som egentlig er laget for .NET, det vil si språkene C#, Visual Basic og J#. Ved å bruke $NET.addAssembly()$ kan .NET funksjoner brukes direkte i MATLAB.

Funksjonene *abb.get*, *abb.set* og *abb.isequal* bruker fire datatyper fra Rapid, *robtarget*, *pose*, *pos* og *num*. Variablene kan settes som en vektor med verdier på samme måte som i Rapid ([[x,y,z],[q1,q2,q3,q4]]), eller som en del av en struktur. Andre typer variabler kan settes ved å sette de sammen av *num* variabler og sette de individuelt.

For å verifisere data er det brukt en sammenligning med toleranse, $abs(a - b) < toleranse$, denne sammenligningen blir brukt for alle verdier i vektoren eller strukturen. Toleransen er brukt fordi det bare blir sendt 7 totale siffer. Dette betyr at 1000.1234 i Rapid blir mottatt som 1000.12 i MATLAB, -1000.1234 blir mottatt som 1000.1.

Programmet i Rapid er skrevet som en tilstandsmaskin, der hver tilstand utfører en spesifikk handling og tilstandene blir styrt direkte fra MATLAB.

Det er tre tooldata variabler i programmet, midtpunktet til *tGripper* er satt 5mm fra enden av gripefingrene, midtpunktet til *tGripperCam* er satt til sentrum av kameralinsen, med kompensering for feil rotasjon i kameraet. *tGripperGrippingPos* er identisk til *tGripper* men er rotert ca 30 grader, slik at de fjærede gripefingrene fjærer dersom de treffer bordet. Rotasjonen til *tGripperGrippingPos* ble funnet ved å rotere i frihånd og lese *currentPos* og *testPos* med
$abb.get(currentPos, 1)$.Prosessen for å finne *tGripperCam* er beskrevet i kapittel 2.1.

De to robtarget variablene *nullPos* og *zero* har begge posisjonen X= 105 og Y= 148.5, noe som tilsvarer sentrum av et A4 ark, dersom det ligger i portrettmodus og origo er øverst til venstre. Det som er ulikt mellom de er at *nullPos* har Z=0 mens *zero* har Z= -400 og at *nullPos* bare har blitt brukt til testing (ved hjelp av 'Go to' på FlexPendanten) mens *zero* blir brukt for å flytte kameraet til sentrum av arket.

For å lage arbeidsobjektet brukes robtarget $p1$, $p2$ og $p3$ samt pos $displ1$, $displ2$ og $displ2$. Robtarget variablene settes til en relativ posisjon i forhold til gjeldene posisjon med for eksempel
$p1 := RelTool(currentPos.displ1.x, displ1.y, displ1.z)$; Displ variablene settes gjennom MATLAB til hjørner og sentrum av arket for så å bruke dette for å konstruere et arbeidsobjekt med $DefFrame(p1, p2, p3)$. Denne funksjonen genererer en *pose* ved å la $p1$ være origo, $p2$ være et punkt på X-aksen og $p3$ være et punkt i XY-planet.

I tillegg til displ variablene, setter MATLAB *matlabPos* og *rotation*, som til sammen blir en robtarget posisjon med rotasjon om Z-aksen. Ved kun å bruke en variabel til bevegelse begrenses feilkilden. Rotasjonen til *matlabPos* blir satt med $OrientZYX(rotation, 0, 0)$.

Programmet kjører kontinuerlig selv om det er satt i tilstand 0, for idle. I hver iterasjon settes *currentPos* og *testpos*, disse er posisjonen til kameraet *tGripperCam* og *tGripper* respektivt. Programmet sjekker status på *currentState* opp imot en stor if-blokk vist i tabell 4. Etter hver iterasjon settes *currentState* til *nextState*. Dette gjør at dersom programmet er opptatt i en tilstand når *nextState* endres så blir den lagt i en kø til iterasjonen er ferdig.

Tilstand	Beskrivelse
0	Idle.
1	Generer workObject workGenerert.
3	Sett rotasjonen på MATLABPos til verdien av rotation.
4	Flytt tGripperGrippingPos til MATLABPos med MoveL og 'fine' zoning.
5	Flytt tGripperGrippingPos til MATLABPos med MoveJ og 'z200' zoning.
6	Flytter tGripperCam til MATLABPos med MoveL og 'fine' zoning.
7	Åpne griperen.
9	Lukk griperen.
98	Flytt tGripper til zero.
99	Flytt tGripperCam til zero.

Tabell 4: Tilstandene i tilstandsmaskinen.

3.4.1 abb.connect

For å koble til roboten er *abb.connect* implementert. Funksjonen laster inn .NET assemblyene som trengs for kommunikasjon, søker på nettverket etter en robot med riktig IP, logger på denne som Default User og spør om master tilgang. PC SDK funksjonen *ABB.Robotics.Controllers.Controller* kan lage et kontroller objekt fra IP adresse eller et objekt returnert fra *scan.GetControllers*. Det er problematisk å koble til kontrollere som er virtuelle ved å bruke lokal IP adresse. Derfor itererer funksjonen over alle kontrollere funnet ved *scan.GetControllers* og sammenligner ønsket IP adresse med IP adressene som er funnet. Dersom en kontroller med riktig IP blir funnet blir indeksen til kontrolleren lagt til i

en vektor, og i de fleste tilfellene vil det bare eksistere en kontroller på hver IP adresse. Det ofte ønskelig å kunne bruke flere virtuelle kontrollere på samme PC. Derfor har funksjonen muligheten for å velge mellom de forskjellige kontrollerne på samme IP adresse. Dersom det finnes to eller flere verdier i vektoren over kontrollere krever funksjonen to parametre der den andre parameteren velger mellom de forskjellige gyldige kontrollerne.

Tilkoblingen åpnes når en gyldig kontroller er funnet og en streng som inneholder IP, bruker og masterflagg blir vist i MATLAB konsollen for feilsøking.

Funksjonen har to parametre IP og *number*. IP er en streng som inneholder IP adressen til den ønskede kontrolleren og kan være 'localhost' og *number* er indeksen av kontrollere på den IP adressen. Funksjonen returnerer og setter den globale variabelen *controller* til kontroller objektet som blir funnet.

3.4.2 abb.disconnect

Logger av fra kontrolleren og sletter kontroller objektet.

Parameteren *ctrl* refererer til kontrolleren som skal logges av og slettes.

3.4.3 abb.logon

Dersom tilkoblingen blir tapt gjennom $abb.logoff$ eller nettverks problemer kan *abb.logon* brukes i stedet for *abb.connect*. Her blir ikke objektet reinitalisert og *controller.Logon* blir kallet på samme måte som i *abb.connect*.

Det er bare ett parameter *ctrl* som refererer til kontrolleren som skal logges på. Dersom den globale variabelen *controller* er satt og*abb.logon* kalles uten parametre blir $ctrl = controller$.

3.4.4 abb.logoff

Logger av fra kontrolleren, men sletter ikke kontroller objektet.

Parameteren *ctrl* refererer til kontrolleren som skal logges av.

3.4.5 abb.create

Forholdet mellom Rapid og MATLAB går ut på at programmene deler variabler mellom seg slik at dersom en endring av variabelen skjer i ett av programmene så skjer den i begge. Variablene eksisterer i Rapid, med en referanse til de i MATLAB. Denne funksjonen lager en slik referanse og returnerer denne.

Funksjonen har fire parametre, der to er valgfrie. Den første parameteren er *ctrl* og er kontrolleren der variabelen finnes, og *variable* er navnet på variabelen. Parametrene*program* og *module* er valgfrie og blir satt til T_ROB1 og $Module1$ dersom de ikke er oppgitt. Alle parametrene er tekststrenger. Funksjonen returnerer et *ABB.Robotics.Controllers.RapidDomain.RapidData* objekt.

3.4.6 abb.get

Forskjellige variabler i Rapid har ulik struktur i MATLAB, for å gjøre visning av variablene mer intuitivt er *abb.get* skrevet.

Denne funksjonen tar to parametre, *rapidData* og *returnStructure*. Referansen *rapidData* viser til et *ABB.Robotics.Controllers.RapidDomain.RapidData* objekt returnert fra *abb.create*. Dersom *returnStructure* er satt og ulik 0, blir returverdien en struktur isteden for en vektor med alle variablene.

Denne funksjonaliteten er ikke brukt i hovedprogrammet, men ble mye brukt til feilsøking og testing da det er mye lettere å se verdien av *posision.X* isteden for å finne verdien i $[200, 200, -200, 1, 0, 0, 0, 1, -2, 0, 0, 9e9, 9e9, 9e9, 9e9, 9e9, 9e9]$. I tillegg er funksjonen brukt i *abb.isequal*.

3.4.7 abb.set

For å sette verdien til en variabel i Rapid brukes *abb.set*. Dette er gjort for å gjøre endring av verdier mer intuitivt.

Funksjonen har to parametre, *rapidData* og *value*. Der *rapidData* er et *ABB.Robotics.Controllers.RapidDomain.RapidData* objekt som skal settes til verdien *value*. Verdiene kan skrives som en vektor som tilsvarer datatypene i Rapid. I MATLAB blir vektorer i vektorer lagt sammen slik at $[[a, b, c], [d, e, f]]$ blir til $[a, b, c, d, e, f]$, mens mange datatyper i Rapid har denne strukturen. For å gjøre programmet lettere å skrive og feilsøke er *value* formatert på tilsvarende måte som i Rapid. For eksempel settes en *pose* variabel til $[[x, y, z], [q1, q2, q3, q4]]$ i Rapid, for å gjøre det tilsvarende i MATLAB med *abb.set* brukes $abb.set(poseVar, [[x, y, z], [q1, q2, q3, q4]])$. Datatypen kan leses ut fra referansen *rapidData*, og ved å bruke denne datatypen samt lengden på vektoren *value* kan deler av en variabel settes. Det er for eksempel ofte ønskelig å bare sette X,Y og Z koordinatene til en *robtarget* variabel uten å endre på rotasjon eller konfigurasjon. Dette kan gjøres ved å la *value* ha en lengde på 3. For å validere at verdien virkelig er satt blir *abb.isequal* brukt for sammenligning av ønsket verdi og reell verdi hentet med *abb.get*, og det er lagt inn en tidsbegrensning på 5 sekunder. Dersom verdiene er ulike etter tiden er gått vil det resultere i en feilmelding.

Datatype	Lengde av *value*	*value*
num	1	[num]
pos	3	[x,y,z]
pose	3	[x,y,z]
pose	7	[[x,y,z],[q1,q2,q3,q4]]
robtarget	3	[x,y,z]
robtarget	7	[[x,y,z],[q1,q2,q3,q4]]
robtarget	17	[[x,y,z], [q1,q2,q3,q4], [cf1,cf4,cf6,cfx], [eax_a, eax_b, eax_c, eax_d, eax_e, eax_f]]

Tabell 5: Forholdet mellom datatyper og lengde av *value*.

Variabelen *value* kan også være en struktur på samme måte som i *abb.get*. Her følges de samme reglene som vist i tabell 5.

3.4.8 abb.isequal

I MATLAB finnes det mange måter å sammenligne to variabler på, vanlig relasjon ('=='), funksjoner som *isequal*, *isequalwitheaqualnans*, *eq* med mer. Ingen av disse kan brukes til å sammenligne to vektorer eller strukturer med en ønsket toleranse. Forsøk har vist at i kommunikasjonen mellom Rapid og MATLAB blir det bare brukt seks siffer inkludert punktum. Det vil si at dersom en variabel blir satt til 1234.56789 i Rapid så vil den avrundes til 1234.57 å ha blitt overført til MATLAB. Hadde dette vært på grunn av visningen i MATLAB, med forskjellige formaterings stiler så hadde det ikke vært et problem. Men 1234.56789 er ulik 1234.57.

Funksjonen *abb.isequal* er til for å løse denne problematikken. Her blir to nummer, vektorer eller strukturer sammenlignet med en toleranse avhengig av størrelsen på tallene. Sammenligningen blir gjort ved å se om absoluttverdien av differansen på alle verdiene er mindre enn toleransen. Det vil si $abs(a - b) < toleranse$. Toleransen blir funnet ved å finne den største absoluttverdien av de to tallene som blir sammenlignet og så gi en verdi avhengig av lengden til tallet. Tall over 1000 får en toleranse på 0.01, tall over 100 får en toleranse på 0.001, tall over 10 får en toleranse på 0.0001, ellers får tallene toleransen 0.00001. Dersom et tall er negativt ('-' tar opp et ekstra siffer), blir toleransen ganget med 10, slik at 0.01 blir til 0.1.

Strukturer og vektorer blir iterert over, og dersom en ulikhet utenfor toleransen blir funnet så blir returverdien satt til 0, ellers blir den satt til 1. For strukturer blir navnene på variablene sammenlignet først. For både strukturer og vektorer blir lengdene sammenlignet. Funksjonen er bare brukt til sammenligning av verdiene i *abb.set*, men kan brukes til sammenligning utenom dette også.

Funksjonen tar to parametre, a og b som kan være tall, vektorer eller strukturer. Dersom disse to er like blir 1 returnert, dersom de er ulike blir 0 returnert.

3.4.9 abb.setState

Rapid programmet er en tilstandmaskin, det er ønskelig å kontrollere denne mer intuitivt enn bare ved *abb.set*. Derfor brukes *abb.setState* til å endre tilstanden, funksjonen prøver å sette nåværende tilstand til ønsket tilstand og gir ut en feilmelding om dette ikke kunne oppnås innen en gitt tid (600 sekunder i programmet). Fordelen med dette er at MATLAB programmet venter til Rapid programmet er ferdig med sin tilstand før neste steg i MATLAB utføres.

Funksjonen har tre parametre, *currentState*, *wantedState* og *value*, der *currentState* og *wantedState* er *RapidData* referanser og *value* er ønsket verdi. Denne verdien settes til *wantedState* og programmet venter til verdien blir satt ved å bruke en while løkke som sammenligner *value* med *currentState*.

4 Resultater

Oppgaven var å detektere klosser og plukke de opp ved hjelp av en ABB robot og et kamera. Utfordringene med dette kom på uventede områder som feil i kameraet eller problemer med kommunikasjonen. Det egentlige resultatet blir koden som er produsert i løpet av perioden som utfører oppgaven.

4.1 Eksempler

For å bruke programmet videre er det ofte greit med eksempler som viser noe av funksjonaliteten. For disse eksemplene er ikke noe kode i funksjonene endret.

4.1.1 Abb kommunikasjon

For å kommunisere mellom MATLAB og ABB roboten er en rekke funksjoner skrevet. Kommunikasjonen fungerer ved å la MATLAB skrive til variabler i Rapid programmet. Først må et kontroller objekt lages.

```
%lager et kontroller objekt med tilknyttning til kontrolleren
%på 10.0.0.1
controller = abb.connect('10.0.0.1');
```

Dette lager .NET assembly referanser i MATLAB og tillater å bruke funksjoner fra disse direkte i MATLAB.

Deretter lages referanser til variablene som skal endres.

```
%variabel av type num
integer = abb.create(controller,'integer')
%variabel av type pos
position = abb.create(controller,'position')
%variabel av type robtarget
target_10 = abb.create(controller,'target_10')
```

Etter at referansene er opprettet kan variablene endres.

```
%setter variabelen integer til 15
abb.set(integer,15);
%setter pos til [10,20,30]
abb.set(position,[10,20,30])
%setter posisjonen til [10,20,30] samt ønsket konfigurasjon av
%armen til [1,-1,0,0]. Rotasjonen er lik som i eksempelet over.
abb.set(target_10,[[10,20,30],[1,0,0,0],[1,-1,0,0],...
         [9e9,9e9,9e9,9e9,9e9,9e9]]
```

Variabler kan selvfølgelig også leses.

```
%verdien av a settes til verdien av integer
a = abb.get(integer);
%verdien av integer er uendret, mens a er øket med 5
a = a+5;
```

Det er også mulighet for å benytte strukturer i set og get.

```
%ved å sette andre parameteren til 1, blir strukturmodus valgt
>>pos = abb.get(position,1);
>>pos

pos =

    X: 10
    Y: 20
    Z: 30

%bare den lokale kopien av strukturen blir endret
>>pos.X = 40;
>>abb.set(position,pos);
>>abb.get(position,1)

ans =

    X: 40
    Y: 20
    Z: 30
```

Når kommunikasjonen er ferdig kan man logge av kontrolleren.

```
abb.logoff(controller)
```

eller koble fra kontrolleren (dette slettet kontroller objektet i tillegg til å logge av)

```
abb.disconnect(controller)
```

4.1.2 Fargesegmentering

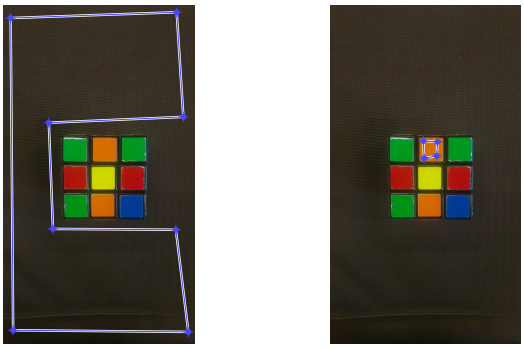
Det første som trengs for å segmentere et bilde er selve bildet. Dette hentes ved å bruke et webcamera.

```
%Henter et bilde I fra webcamera 1 med forhåndsvisning
I = getFrame(1,1);
```

For eksempelets skyld er et fargerikt objekt benyttet. Nå må fargene defineres til videre klassifisering. getColorMarkers vil la brukeren velge områdene

som inneholder de forskjellige fargene, mens navnet på fargen vises i konsollvinduet. Navnene blir hentet fra *names* variabelen og kan brukes videre i andre funksjoner.

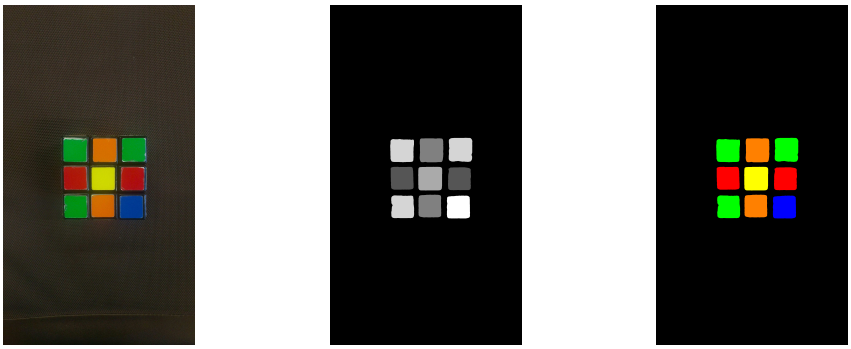
```
%Navnene til fargene for bedre tilbakemelding fra programmet
names = {'Bakckground', 'White','Red','Orange',...
         'Yellow','Green','Blue'};
%Antall farger er det samme som lengden av names
colorMarkers = getColorMarkers(I,size(names,2),names);
```



Figur 23: Eksempel av fargevalg med getColorMarkers.

Deretter er det lurt å reklassifisere bildet, det vil si å bruke klassifiseringen hentet på bildet på seg selv. Dette gjør det lettere å bli oppmerksom på feilklassifisering med en gang.

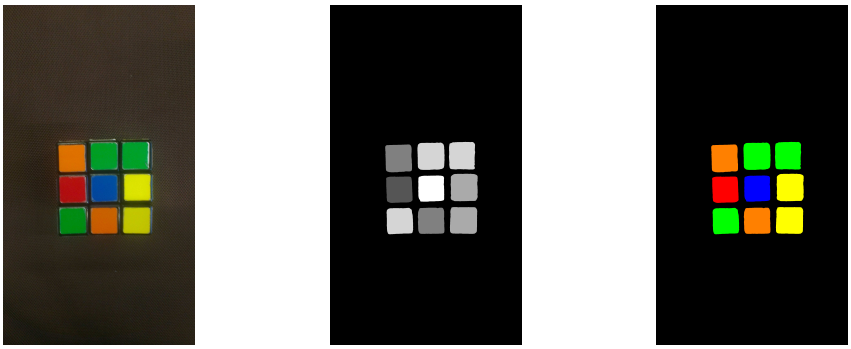
```
[images,combinedImage] = getSegmentedImage(I,colorMarkers);
imshow(combinedImage,[]);
```



Figur 24: Reklassifisering av bildet. Figuren til høyre er fargelagt for lettere å illustrere klassifiseringen.

Fra reklassifiseringen er det tydelig å se seks gråtonenyanser som stemmer med det originale bildet. Det fargelagte bildet er nesten identisk til originalen, det er i alle fall mulig å skille de forskjellige rutene i riktig farge. Et nytt bilde av Rubik's kuben blir tatt for å teste klassifiseringen.

```
I_ny = getFrame(1,1);
[images_ny,combinedImage_ny] = getSegmentedImage(I_ny,colorMarkers);
imshow(combinedImage,[]);
```



Figur 25: Klassifisering av et nytt bilde med samme colorMarkers. Figuren til høyre er fargelagt for lettere å illustrere klassifiseringen.

Det er tydelig at klassifiseringen har fungert i dette tilfellet også. Fargeleggingen ble gjort i MATLAB ved bruk av følgende kode.

```
c = zeros(size(I));
r = c(:,:,1);
g = c(:,:,2);
b = c(:,:,3);
%rød = (255,0,0)
r(combinedImage == 2) = 255/255;
g(combinedImage == 2) = 0;
b(combinedImage == 2) = 0;
%oransje = (255,128,0)
r(combinedImage == 3) = 255/255;
g(combinedImage == 3) = 128/255;
b(combinedImage == 3) = 0;
%gul = (255,255,0)
r(combinedImage == 4) = 255/255;
g(combinedImage == 4) = 255/255;
b(combinedImage == 4) = 0;
%grønn = (0,255,0)
r(combinedImage == 5) = 0;
g(combinedImage == 5) = 255/255;
b(combinedImage == 5) = 0;
%blå = (0,0,255)
r(combinedImage == 6) = 0;
g(combinedImage == 6) = 0;
b(combinedImage == 6) = 255/255;
c(:,:,1) = r;
c(:,:,2) = g;
c(:,:,3) = b;
```

Ved å bruke *getCornersFromImages* kan sentrum i de individuelle rutene finnes. Gjennom dette kan man få ut en matrise med fargene som kan brukes videre i løsningen av kuben.

```
%returnerer en struktur med farger, sentrum og hjørnepunkter
%til alle objektene
col = getCornersFromImages(images,names);

%lager en ny, flatere struktur som passer bedre til rubikskube
countCube=1;
for countCol = 1:length(col)
    for countObject = 1:length(col(countCol).object)
        center = col(countCol).object(countObject).center
        cube(countCube).pos = center;
        cube(countCube).color = col(countCol).name;
        countCube = countCube +1;
    end
end
```

For å ha et overblikk over posisjonene blir maksimal og minimal posisjon av sentrene i hver retning funnet.

```
%for løkke for å finne maksimal og minimal posisjon i hver retning
maxX = 0; minX = 2000;
maxY = 0; minY = 2000;
for cubeCount = 1:length(cube)
    position = cube(cubeCount).pos
    if position(1) > maxX
        maxX = position(1);
    end
    if position(1) < minX
        minX = position(1);
    end
    if position(2) > maxY
        maxY = position(2);
    end
    if position(2) < minY
        minY = position(2);
    end
end
```

Deretter får den nye *cube* strukturen navn som viser til posisjonen av rutene avhengig av sentrum av hver rute.

```
%Posisjonene blir navngitt avhengig av posisjon samt at
%cellArrayen cubeMatrix blir oppdatert med navnene
cubeMatrix = {0,0,0;0,0,0;0,0,0}
for cubeCount = 1:length(cube)
    position = cube(cubeCount).pos
    if abs(position(2) - maxY)< maxY/10
        cube(cubeCount).Ypos = 'bottom';
        Ypos = 3;
    elseif abs(position(2) - minY)< minY/10
        cube(cubeCount).Ypos = 'top';
        Ypos = 1;
    else
        cube(cubeCount).Ypos = 'middle';
        Ypos = 2;
    end
    if abs(position(1) - maxX)< maxX/10
        cube(cubeCount).Xpos = 'right';
        Xpos = 3;
    elseif abs(position(1) - minX)< minX/10
        cube(cubeCount).Xpos = 'left';
        Xpos = 1;
    else
        cube(cubeCount).Xpos = 'center';
        Xpos = 2;
    end
    cubeMatrix{Ypos,Xpos} = cube(cubeCount).color
end
```

Resultatet blir en struktur med navn og posisjoner i koordinater og rutenett, pluss en cell array med navnene i rutene. Begge disse to kan enkelt brukes videre i løsning av kuben.

pos	name	Ypos	Xpos
[666 1776]	'Red'	'middle'	'left'
[661 1476]	'Orange'	'top'	'left'
[974 2076]	'Orange'	'bottom'	'center'
[1283 2071]	'Yellow'	'bottom'	'right'
[1289 1755]	'Yellow'	'middle'	'right'
[672 2076]	'Green'	'bottom'	'left'
[972 1455]	'Green'	'top'	'center'
[1270 1452]	'Green'	'top'	'right'
[973 1764]	'Blue'	'middle'	'center'

Tabell 6: Innholdet i *cube* strukturen.

```
1
2  cubeMatrix =
3      'Orange'    'Green'     'Green'
4      'Red'       'Blue'      'Yellow'
5      'Green'     'Orange'    'Yellow'
```

4.2 Fullskala gjennomkjøring av programmet

MATLAB programmet er delt opp i blokker ved å bruke '%%'. Dette gjør at individuelle deler av programmet kan kjøres uavhengig av hverandre.

Det første som må gjøres i programmet er å åpne filen med kalibreringsdata eller lage ny kalibreringsdata.

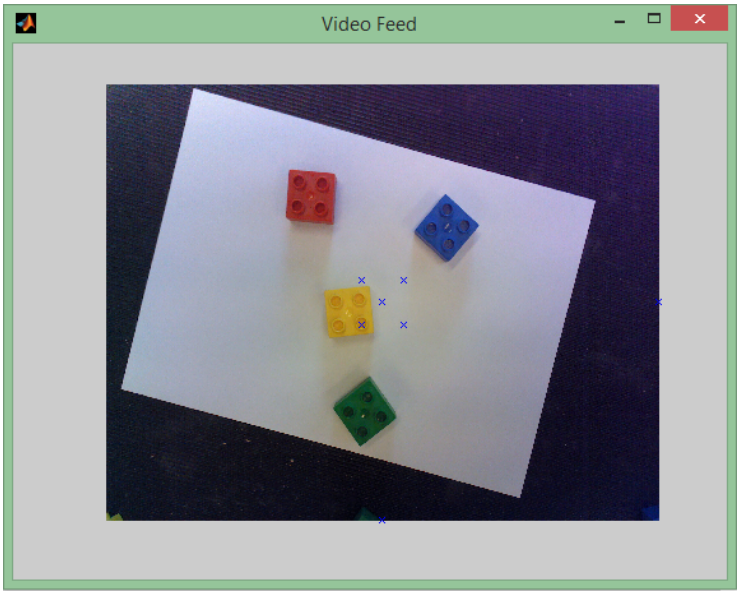
```
1  %% Calibration
2  %  Calibration of extrinsic parameters.
3  load('calib\calibration.mat');
4  global controller
5  if ~exist('tform')
6      I = getFrame(2);
7      tform = getTransform(I,25,cameraParameters);
8  end
```

Variabelen *controller* blir gjort global for å kunne få tilgang til den i *abb.set* dersom kommunikasjonen skulle feile.

Roboten må deretter jogges slik at arket er synlig. Arket trenger ikke ha riktig vinkel, men alle fire hjørner må være synlige. Funksjonen *getFrame* med 1 som andre parameter gjør det mulig å 'sikte' gjennom kameraet. Når arket er synlig trykker brukeren på en knapp og programmet går videre.

```
%% Creation of work object
%  In order to create a work object, a piece of paper
%  is used to determine the natural work object
%  according to the paper.

% This is usually where you would align the
% camera to actually see the paper
I = getFrame(2,1);
```



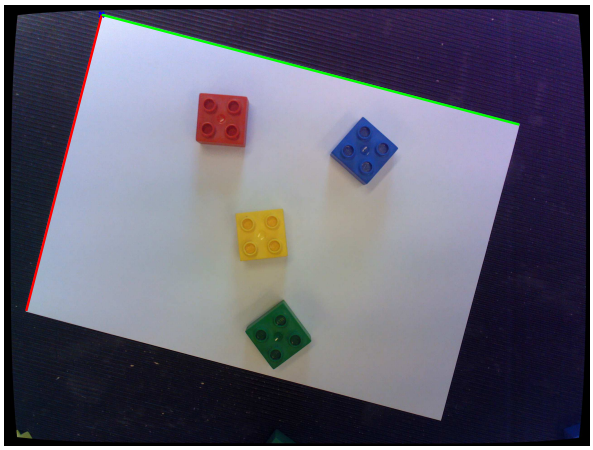
Figur 26: Video direkte fra kameraet. Kryssene i midten viser bildesentrum og er laget for bruk i kalibrering. Bildet er også behandlet for å redusere radial forvrengning.

Deretter må dette bildet behandles for å finne hjørnene ved hjelp av *getCornersPaper*.

```
I = undistortImage(I,cameraParameters);
[corners, center, paper] = getCornersPaper(I);
```

Den relative posisjonen til hjørnene blir funnet som vist i figur 22. Origo, et punkt langs X-aksen og et punkt langs Y-aksen blir funnet (*origo*, *pointX*, *pointY*). Deretter blir aksene plottet over bildet.

```
1
2  I = undistortImage(I,cameraParameters);
3  [corners, center, paper] = getCornersPaper(I);
4  figure,imshow(I),
5  title('Coordinate system of defined workObject');
6
7  imageCenter = [size(I,2),size(I,1)]./2;
8  for j=1:length(corners)
9      distance(j) = (corners(j,1).^2+corners(j,2).^2).^.5;
10 end
11 [value,index] = sort(distance);
12 origo = corners(index(1),:);
13 distanceToOrigo(1) = ((origo(1)-corners(index(2),1))^2 ...
14                     + (origo(2)-corners(index(2),2))^2).^.5;
15 distanceToOrigo(2) = ((origo(1)-corners(index(3),1))^2 ...
16                     + (origo(2)-corners(index(3),2))^2).^.5;
17 [valueO,indexO] =sort(distanceToOrigo);
18 pointX = corners(index(indexO(1)+1),:);
19 pointY = corners(index(indexO(2)+1),:);
20 hold on
21 plot(origo(1),origo(2),'bx','LineWidth',2);
22 plot([origo(1),pointX(1)],[origo(2),pointX(2)],'r','LineWidth',2);
23 plot([origo(1),pointY(1)],[origo(2),pointY(2)],'g','LineWidth',2);
```



Figur 27: Akser plottet over et bilde av arket. Den røde er X-aksen, mens den grønne er Y-aksen.

Til slutt blir distansen til *origo*,*pointX* og *pointY* regnet ut i verdensko-

ordinater. Vinkelen mellom X-aksen og den vertikale aksen til bildet blir også regnet ut for at brukeren skal kunne kontrollere treffsikkerheten. I dette tilfellet er vinkelen -14.2401º.

```
[dxOrigo, dyOrigo]   = getWorldDist(imageCenter,origo,tform);
[dxPointX,dyPointX]  = getWorldDist(imageCenter,pointX,tform);
[dxPointY,dyPointY]  = getWorldDist(imageCenter,center,tform);
%Angle between camera and paper, for verification.
angle =atan2(dyPointX-dyOrigo,dxPointX-dxOrigo)*180/pi
```

I neste programblokk blir alle $dx---$ og $dy---$variablene sendt til roboten for å lage arbeidsobjektet. Først må kontroller objektet kobles til roboten.

```
%% Initiate ABB robot and create work object
controller = abb.connect('152.94.0.38');
```

Deretter blir alle referansene til variabler i rapid opprettet.

```
currentState = abb.create(controller,'currentState');
nextState = abb.create(controller,'nextState');
displ1 = abb.create(controller,'displ1');
displ2 = abb.create(controller,'displ2');
displ3 = abb.create(controller,'displ3');
bord = abb.create(controller,'bord');
rotation = abb.create(controller,'rotation');
MATLABPos = abb.create(controller,'MATLABPos');
currentPos = abb.create(controller,'currentPos');
abb.setState(currentState,nextState,0);
```

Avstanden til bordet (i forhold til wobj0) blir beregnet og $disp1$-$disp3$ blir satt. Disse posisjonene i rommet tilsvarer $origo$, $pointX$ og $pointY$ i bildet.

```
distToTable = abb.get(currentPos,1).Z-abb.get(bord);
abb.set(displ1,[dxOrigo, dyOrigo,distToTable]);
abb.set(displ2,[dxPointX,dyPointX,distToTable]);
abb.set(displ3,[dxPointY,dyPointY,distToTable]);
```

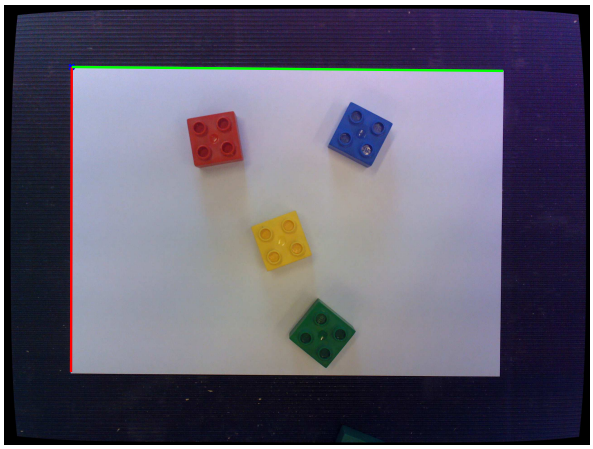
Rapidprogrammet behandler disse punktene for å lage arbeidsobjektet $workGenerert$.

```
abb.setState(currentState,nextState,1);
abb.setState(currentState,nextState,99);
abb.setState(currentState,nextState,0);
abb.logoff(controller);
```

Arbeidsobjektet blir generer når $currentState = 1$. Det er bare $uframe$ delen som blir endret, resten er som i wobj0. Når $currentstate = 99$ flyttes kameraet til sentrum av arket.

```
1  p1 := RelTool (currentPos,  displ1.x,  displ1.y , displ1.z);
2  p2 := RelTool (currentPos,  displ2.x , displ2.y , displ2.z);
3  p3 := RelTool (currentPos,  displ3.x,  displ3.y , displ3.z);
4  frame := DefFrame(p1,p2,p3);
5  workGenerert.uframe := frame;
```

Ved å kjøre forrige programblokk kan resultatet verifiseres. Vinkelen i forhold til kameraaksene er nå -0.1266^{o}.

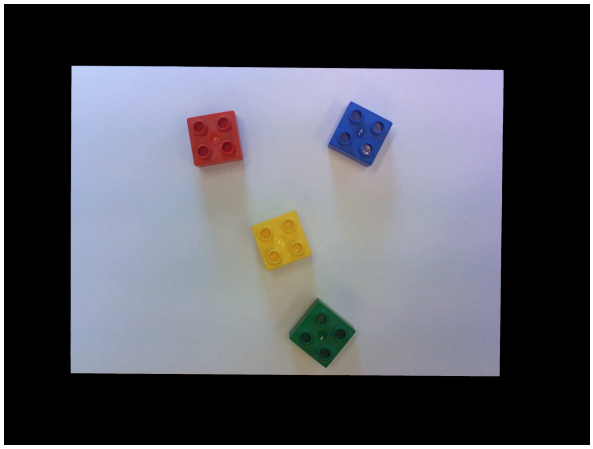


Figur 28: Akser plottet over et bilde av arket etter å ha generert arbeidsobjektet og flyttet kameraet til sentrum. Kameraet er også rotert slik at aksene til arket er vinkelrett på kameraaksene. Den røde er X-aksen, mens den grønne er Y-aksen.

Når arbeidsobjektet først er generert, trengs ikke ny generering før arket flyttes på.

Alt er nå klart for å segmentere, detektere og plukke klosser. Først må kontrolleren logges på og kameraet flyttes til sentrum før et nytt bilde blir tatt. Sentrum her, som alle andre punkter i rommet er avhenig av arbeidsobjektet som ble generert tidligere. Et tidligere sett med color_markes fra *getColorMarkers* blir definert og innholdet på arket blir funnet ved *getCornersPaper*. Samtidig finnes hjørnene til arket.

```
%% Actual picking up part abb.logon(controller);
abb.setState(currentState,nextState,99);
I = getFrame(2);
I = undistortImage(I,cameraParameters);
color_markers = [
    128.5905  122.3360; %background
    164.3418  143.7367; %red
    133.7915  181.5070; %yellow
    113.3183  115.9544; %green
    138.1641   90.3943; %blues
];
names = {'Background','Red','Yellow','Green','Blue'};
[corners,center,paper] = getCornersPaper(I);
```



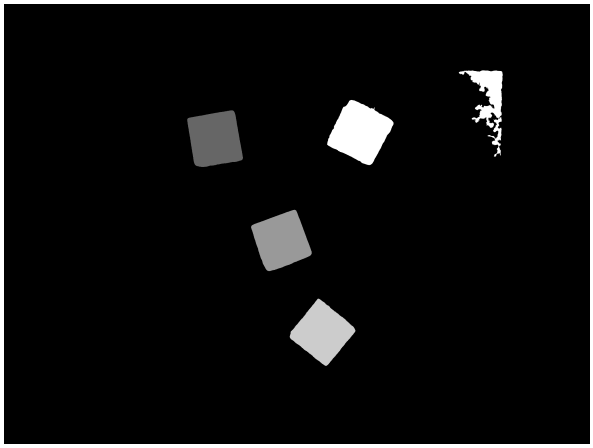
Figur 29: Innholdet av arket med alt utenfor fjernet.

Origo på arket som ble brukt i genereringen finnes på nytt. Ved å måle avstander til origo i bildet (i verdenskoordinater) kan koordinatene i arbeidsobjektet finnes.

```
imageCenter = [size(I,2),size(I,1)]./2;
clear distance;
for j=1:length(corners)
    distance(j) = (corners(j,1).^2+corners(j,2).^2).^.5;
end
[value,index] = sort(distance);
origo = corners(index(1),:);
```

Selve segmenteringen blir så gjort. Til dette brukes funksjonen *getSegmentedImage*.

```
[images,combinedImage] = getSegmentedImage(paper,color_markers);
```



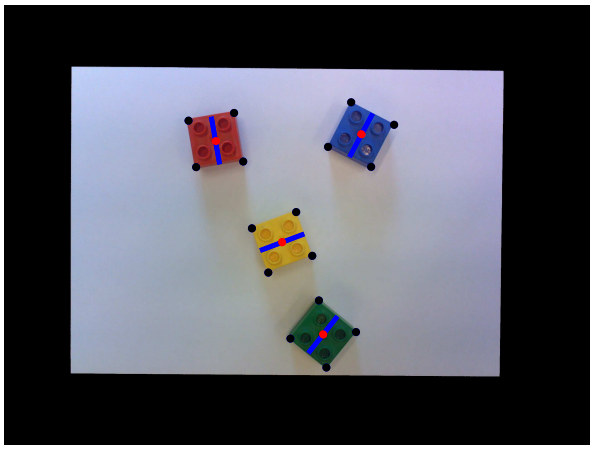
Figur 30: Kombinerte segmenter fra fargesegmentering (*combinedImage*), det er noe støy fra områder av arket som blir klassifisert som blåe.

Så kommer hjørnedeteksjon og sentrum fra *getCornersFromImage*. Dette blir lagret i *col* strukturen.

```
col = getCornersFromImages(images,names);
```

Videre brukes sidepunktene vist i figur 22 (her kalt p1 og p2) for å regne ut vinklene til alle objektene til alle fargene. Vinklene blir korrigert med -180º dersom de er over 90º. På den måten begrenses området griperen må bevege seg i og kameraet er alltid på utsiden av griperen slik at det ikke blir knust mellom griperen og armen. Til slutt blir vinklene lagret i *col* strukturen.

```
for colorCount = 1:length(col)
    for objectCount = 1:length(col(colorCount).object)
        X = col(colorCount).object(objectCount).corners(:,1);
        Y = col(colorCount).object(objectCount).corners(:,2);
        center =col(colorCount).object(objectCount).center;
        x1 = X(1);
        y1 = Y(1);
        distance = ((x1 - X).^2 + (y1 -Y).^2).^.5;
        if length(distance) == 4
            [value,index] = sort(distance);
            %     x1--------------x2
            %     |               |
            %     p1      c       p2
            %     |               |
            %     x3--------------x4
            x2 = X(index(3));
            y2 = Y(index(3));
            x3 = X(index(2));
            y3 = Y(index(2));
            x4 = X(index(4));
            y4 = Y(index(4));
            p1 = [(x1+x3)/2 (y1+y3)/2];
            p2 = [(x2+x4)/2 (y2+y4)/2];
            angle =atan2(p2(1)-p1(1),p2(2)-p1(2))*180/pi;
            if angle > 90
                angle = angle -180;
            end
            col(colorCount).object(objectCount).angle = angle;
        end
    end
end
```



Figur 31: Innholdet i *col* strukturen plottet på bildet. Svarte prikker er hjørner, røde prikker er senterpunkt og den blå linjen illustrerer vinkelen til klossen. Når klossen blir grepet vil det skje vinkelrett på den blå linjen.

I neste blokk åpnes griperen å sette *currentState* til 7 og selve plukkingen begynner. Dette er en stor løkke med mye gjentakelse og bare hovedelementene er tatt med her.

```
1  abb.setState(currentState,nextState,7);
```

For å finne senterkoordinatene til hver kloss brukes *getWorldDist*

```
1  [dxKloss, dyKloss]    = getWorldDist(origo,center,tform);
```

Følgende blir gjentatt for hvert punkt roboten flyttes til. I første linje settes posisjonen roboten skal flyttes til. I andre og tredje linje settes rotasjonen. Fjerde linje utfører selve bevegelsen, dersom tilstanden settes til 4 brukes en nøyaktig og lineær bevegelse, mens dersom den blir satt til 5 brukes en rask og ulineær bevegelse.

```
1  abb.set(MATLABPos,[[dxKloss,dyKloss,-200],[1,0,0,0],...
2          configuration,[9E9,9E9,9E9,9E9,9E9,9E9]]);
3  abb.set(rotation,angle);
4  abb.setState(currentState,nextState,3);
5  abb.setState(currentState,nextState,5);
6  abb.setState(currentState,nextState,0);
```

4.3 Tooldata til kameraet

To uker med prøving, feiling, måling og korrigering gjorde til slutt at en*tooldata* for kameraet ble funnet:

tGripperCam:=[TRUE, [[53.69 ,0 ,41.19], [0.997969, 0.00541994, 0.0171534, -0.0611147]], [1, [0, 0, 1], [1, 0, 0, 0], 0, 0, 0]]; Denne peker på sentrum i kameralinsen med riktig rotasjon og viser at sentrum er 53.69 mm forskjøvet i X akse og 41.19 mm forskjøvet i Z akse (i forhold til festepunktet for verktøyet). Rotasjonen er uttrykt i quaternioner og representerer -7^{o} om Z-aksen, 2^{o} om X-aksen og 0.5^{o} om Y-aksen.

4.4 Forsøk

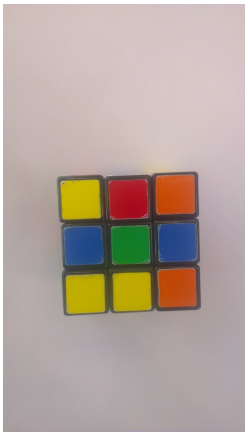
Forskjellige deler av prosessen har ulik tidsforbruk, ved å måle tiden i hver del skapes et bedre inntrykk av hvor det ligger et forbedringspotensiale. I tillegg har en alternativ løsning til fargesegmenteringen blitt benyttet.

4.4.1 Sammenligning mellom min-dist og maximum likelihood.

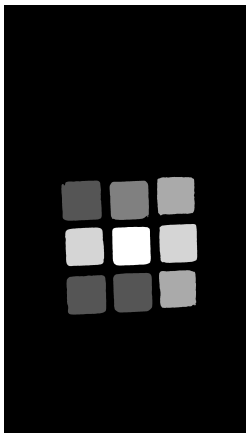
En maximum likelihood implementasjon (funnet i [8]) blir benyttet for å klassifisere fargene, på samme måte som minimal distanse algoritmen.

	Minimal distanse	Maximum Likelihood
Gjennomsnitt tid	6.3506 sekunder	731.8309 sekunder

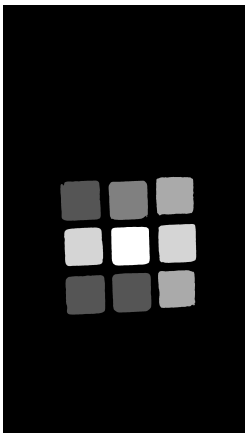
Tabell 7: Sammenligning av tidsforbruk.



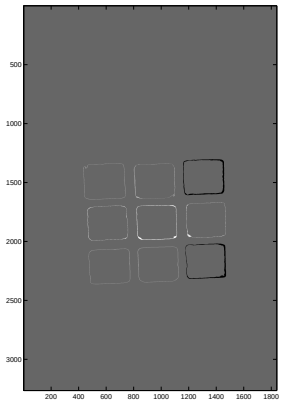
Original



Minimal distanse



Maximum Likelihood



Differanse.

Figur 32: Forskjellen i segmentering ved bruk av minimal distanse (getSegmentedImage) og maximum likelihood.

4.4.2 Tidsforbruk i ulike deler av programmet

Testene er utført med seks farger på et 3264x1840px bilde. Funksjonene *getSegmentedImage* og *getCornersFromImages* er testet siden disse er mest kritiske dersom man ønsker raskere prosessering. Kalibrering av kameraet eller lignende er ikke nødvendig å ta tiden på siden de ikke trenger å bli kjørt i hver iterasjon. Et gjennomsnitt av 10 forsøk er brukt.

Beskrivelse	Kode	Tidsforbruk (i sekunder)
Hele getCornersFromImages (Avhengig av antall farger, og objekter)	**getCornersFromImages(im);**	**1.6390021**
Boundary trace i getCornersFromImages	bwboundaries(im(:,:,count));	0.2173133
hjørnedeteksjonen i getCornersFromImages	getCorners(BW_segment);	0.0531148
Hele getSegmentedImage	**getSegmentedImage(I,colorMarkers);**	**6.2540630**
Konvertering til L*A*B*	cform = makecform('srgb2lab'); lab_I = applycform(I,cform);	3.6444617
Måling av distansene i fargerommet.	((a - colorMarkers(count,1)).^2 + (b - colorMarkers(count,2)).^2).^.5;	0.3279858
Binær bildebehandling	imclearborder bwareaopen bwfill	1.7911580

Tabell 8: Forskjellen i tidsforbruk ved de individuelle delene av programmet.

4.5 Videoer

Første video viser testing med bare linæere bevegelser.

`https://www.youtube.com/watch?v=QAApV1T90jY`

Andre video viser generering av arbeidsobjektet og plukking med lineære og ulinæere bevegelser.

`https://www.youtube.com/watch?v=6z6TUfYxd7w`

Tredje video viser mange forskjellige oppsett av klosser. Plukkingen er lik som i den andre videoen. I slutten vises også lineær jogg i forhold til arbeidsobjektet.

`https://www.youtube.com/watch?v=nOADyen2cOE`

4.6 Utfordringer og begrensinger

Det største problemet i hele prosessen har vært kameraet. Kamera sensoren ligger skeivt i forhold til kamerahuset, noe som gjør at all data fra kameraet er feil i forhold til forventet data. Dette er veldig frustrerende når det ser riktig ut fra kameraet, men blir feil når kameraet blir flyttet siden arbeidsobjektet er feil i forhold til bildet. Senere i prosessen ble lineær jogg langs aksene brukt for å verifisere abeidsobjektet og det gjorde feilsøkingen og korrigeringen litt lettere til løsningen beskrevet i kapittel 2.4.4 ble funnet.

Lysfølsomhet har også vært et problem med kameraet. Deteksjonen av fargene er avhengig av lysforholdene, noe som gjør at gjennomsnittene for de kjente

fargene (colorMarkers) må redefineres i forskjellige lys. Vignetting gjør farger i periferien av bildet mørkere, det gjør at farger ser ulike ut i forhold til distansen fra sentrum. Gule klosser er spesielt preget av dette og blir ofte ikke gjenkjent dersom de er plassert i periferien av bildet.

Figur 33: Fargeforskjell i forhold til distanse fra sentrum i bildet. De gule klossene i venstre til bildet har en annen farge enn klossen i midten. Under er klossene vist ved siden av hverandre.

Refleksjoner fra lyset i taket er et beslektet problem, det gjorde at områder av klossene ble hvite fra refleksjonen. Dette ga hull i segmenteringen og manglene hjørner. Ved å pusse klossene med fint sandpapir ble de matte og problemet var løst.

Figur 34: Upusset (øverst) og pusset (nederst) kloss.

Det har også vært problemer med kommunikasjonen mellom MATLAB og Rapid også. MATLAB mister master tilgang dersom der oppstår problemer eller avbrudd i kommunikasjonen, dette har det ikke blitt noe løsning på annet enn å spørre etter master tilgang på nytt. Et annet problem i kommunikasjonen er at det virker som variabler bare blir sendt som 7 siffer inklusivt punktum og fortegn. Dette er ikke et stort problem, men kan være frustrerende.

L*A*B* og HSV/HSL fargerom bruker en variabel for lys styrke(light, lightness, value, etc) det betyr at hvit og svart bare er skilt med denne variabelen. Dersom lysstyrken blir fjernet som det er gjort i fargeklassifiseringen her, blir det umulig å se forskjell på hvit og svart. Denne variabelen er fjernet for å få raskere klassifisering av sterke farger på bekostning av gråtoner.

4.7 Diskusjon

Fargerommet L*A*B* ble valgt fordi det er en tilnærmelse av den menneskelige fargeforståelsen. Det er fult mulig å bruke andre fargerom som RGB eller HSV eller lignende,.Dersom det skal brukes et annet fargerom trengs en annen måte å klassifisere fargene. Maximum likelihood er et godt alternativ, men krever mer prosessorkraft. Et annet alternativ er å bruke kantdeteksjon for å finne kanter i et gråskala bilde og så klassifisere områdene innenfor kantene som blir funnet. Siden det da bare er nødvendig å klassifisere en mindre del av bilde blir prosesseringstiden kuttet ned drastisk. Det er også mulig å nedsample bildet for å gi raskere klassifisering, ved å nedsample bildet med en faktor på 0.5 (50% av original størrelse) går prosesseringstiden ned med 75%, men det krever at bildet har en viss oppløsning for å få riktig klassifisering.

Personlig er jeg ikke en tilhenger av hvordan kommunikasjonen foregår mellom MATLAB og Rapid. Det å dele variabler direkte uten noe intern verifisering er litt skummelt. Derfor er det laget funksjoner som setter variabler med verifisering og feilhåndtering. Det burde vært en mulighet å kalle funksjoner i Rapid direkte fra MATLAB der variabler blir passert i kallet, for eksempel:

```
1 abb.call('MoveL [[X,Y,Z],[Q1,Q2,Q3,Q4]]');
```

eller

```
1 abb.moveL([X,Y,Z],[Q1,Q2,Q3,Q4])
```

og det er derfor jeg har valgt å implementere funksjonene etter dette idealet, selv om det nå er nødvendig å sette en robtarget først og en state etterpå istedenfor å kalle en funksjon direkte. Ideelt sett hadde det eksistert en direkte kommunikasjon som ikke går gjennom PC SDK. PC SDK krever en 32 bits MATLAB som begrenser programmet til 4gb ram. Den originale namespacen brukt av PC SDK er rotete og forvirrende, det å bruke $variabel.Value.Value = X$ er lite pragmatisk og jeg hadde store problemer med å holde oversikten i programmet når jeg brukte dette. Etter alle $abb.*$ funksjonene var ferdige var kommunikasjonen mye mer oversiktlig.

Det er tatt en del valg i programmet når det gjelder format. UxVxN formatet for klassifiseringen er valgt fordi jeg ville unngå å bruke cell arrays eller strukturer. Alternativet er det kombinerte bildet med alle klassifiseringene som kan brukes på samme måte og kan mye mindre plass i minnet. For 256 klasser bruker UxVxN $U \cdot V \cdot 256$ bits, mens det kombinerte bildet bare bruker $U \cdot V \cdot 8$bits. Dette kan være en faktor i programmet siden det er en ram begrensning med 32 bits MATLAB.

Det er også mulig å bruke skjermkortet til prosessering i MATLAB, ved å bruke *gpuarray*, men dette er ikke gjort grunnet manglende maskinvare.

På grunn av for kort avstand er det noen ganger mulig at sidene av klossen blir detektert som toppen av klossen. Dette kan unngås ved å bruke et lavere bord eller ta bildet fra høyere oppe. Jeg har valgt å ikke ta stilling til dette

siden det gir lite feilmargin uansett og denne blir korrigert av hvordan klossene plukkes opp som beskrevet i kapittel 3.3.

4.8 Videre arbeid

Programmet og funksjonene til programmet er skrevet for å kunne brukes videre i andre prosjekter. Som vist i eksemplene kan fargesegmenteringen blant annet. brukes til Rubik's kube, men det krever en løsning for hvite ruter. Det er kanskje behov for en bedre måte å segmentere på, for eksempel maximum likelihood, eller bare et annet fargerom til klassifisering. Eventuelt kan segmenteringen gjøres i et gråskala bilde og klassifisering av fargene etterpå.

Det er også deler av oppgaven som kan utvikles direkte, som evnen til å bygge noe med DUPLOen eller evnen til å finne områder ved hjelp av strekkoder eller lignende og lage punkter i Rapid ut ifra disse.

Dersom det ønskes sanntids gjenkjenning av objekter må tidsforbruket begrenses kraftig, dette krever ny klassifisering og endret hjørnedeteksjon.

Funksjonene skrevet kommunikasjon er mye mer oversiktlige enn å arbeide med PC SDK direkte i MATLAB slik at de kan brukes i undervisning eller til videre prosjekter med robotene.

5 Konklusjon

I denne oppgaven har jeg sett nærmere på deteksjon av fargede objekter og kommunikasjon mellom matlab og robot. Hovedkonklusjonen er at programmet fungerer som ønsket, på tross av utfordringer underveis.

Selv om det har vært store utfordringer, fungerer programmet som ønsket. Det er brukt mye tid på feilsøking grunnet feil i kamera og problemer med kommunikasjonen, men disse utfordringene ble løst etter hvert. Koden er skrevet for å kunne brukes videre og kan brukes med enkelte funksjoner eller funksjoner som er laget for å passe sammen. Det er noe rom for videre utvikling av løsningen direkte, for eksempel å kunne bygge med DUPLOen eller å kunne orientere seg ved hjelp av strekkoder.

Et annet hovedfunn er at oversetting av koordinater fra kameraet til roboten fungerer svært bra så lenge det er en kjent referanse i bruk, som i denne oppgaven er arket. Toolboxen fra MATLAB bruker liten tid på å oversette koordinater og for å fjerne forvrengning. Nyeste versjon av MATLAB 2014b, har en egen extrinsic funksjon som gjør hele kamera kalibreringen enda lettere igjen.

Et fremtidig utviklingspotensial ligger i fargesegmenteringen. Fargesegmenteringen er for treg til å brukes i sanntid og har derfor utviklingspotensiale i den retningen. Maximum likelihood kan ikke brukes direkte for å klassifisere farger dersom tid er en faktor. Det som tar lengst tid ved segmenteringen er konverteringen til L*A*B* fargerommet noe som betyr at dersom man ønsker raskere prosessering, så må alternativ til denne konverteringen benyttes. Noe som er verdt å prøve er å segmentere klossene i et gråskala bilde og så klassifisere noen få piksler i senteret av hver kloss.

Det har vært noen utfordringer med gjenkjenne farger, spesielt med gule og blå klosser. Gule klosser er mottagelig for feilklassifisering som bakgrunn (hvit), mens bakgrunnen ofte blir feilklassifisert som blå. Belysning kan være en medvirkende faktor til dette. Denne feilklassifiseringen kan observeres i videoene. Klassifisering av gule og grønne klosser er mest stabil og til demonstrasjoner burde disse brukes.

Treffsikkerheten til hjørnedeteksjonen er avhengig av at segmenteringen er bra og finner alltid de mest ekstreme punktene av objektene dersom segmenteringen gir et bra resultat. Boundary trace er det tregeste elementet av denne prosessen.

Bevegelsene til roboten er nøyaktige i forhold til de detekterte hjørnene. Tooldata til kameraet kan brukes i lignende prosjekter.

Totalt sett er alle målene ved oppgaven utført og resultatet er et program som kan brukes til demonstrasjoner av roboten og bildebehandling, samt et kodebibliotek som kan brukes i undervisning og videre prosjekter.

Referanser

[1] http://www.mathworks.se/help/images/examples/color-based-segmentation-using-the-l-a-b-color-space.html (20.02.14)

[2] http://en.wikipedia.org/wiki/Lab_color_space (20.02.14)

[3] http://www.imageprocessingplace.com/downloads_V3/root_downloads/tutorials/contour_tracing_Abeer_George_Ghuneim/moore.html (28.03.14)

[4] http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html (29.03.14)

[5] http://www.mathworks.se/help/vision/ref/cameraparameters-class.html (21.04.14)

[6] Feature Detection, Extraction, and Matching, MATLAB 2013b dokumentasjon

[7] http://www.mathworks.se/help/images/examples/color-based-segmentation-using-k-means-clustering.html(19.05.14)

[8] Richard O. Duda, Peter E. Hart, David G. Stork. Pattern Classification, second edition. 2001.

[9] http://www.mathworks.se/help/images/ref/multithresh.html(28.05.14)

[10] http://www.mathworks.se/help/images/ref/regionprops.html (28.05.14)

[11] http://www.mathworks.se/help/images/ref/bwboundaries.html(28.05.14)

[12] http://www.mathworks.se/help/images/ref/corner.html (28.05.14)

[13] http://www.mathworks.se/help/images/ref/imclearborder.html(28.05.14)

[14] http://www.mathworks.se/help/images/ref/bwareaopen.html (28.05.14)

[15] http://www.mathworks.se/help/images/ref/imfill.html(28.05.14)

[16] Pierre Soille. Morphological Image Analysis: Principles and Applications. 1999.

[17] David A Forsyth, Jean Ponce. Computer vision: A modern approach, second edition. 2012.

[18] ABB Robotics, RAPID Reference Manual, RAPID Overview On-line (11.06.14)

[19] ABB Robotics, RAPID Reference Manual, System Data Types and Routines On-line (11.06.14)

[20] ABB Robotics, Application manual PC SDK (11.06.14)

A Innhold på CD

- TykhelleMartin.pdf
- Matlabkode
  - Abb
    - connect.m
    - create.m
    - disconnect.m
    - get.m
    - isequal.m
    - logoff.m
    - logon.m
    - set.m
    - setState.m
  - calibration.mat
  - getColorMarkers.m
  - getCorners.m
  - getCornersFromImages.m
  - getCornersPaper.m
  - getFrame.m
  - getSegmentedImage.m
  - getTransform.m
  - getWorldDist.m
  - main.m
- Videoer
  - Video 1
  - Video 2
  - Video 3
- Rapid pack-and-go fil

B MATLAB kode

B.1 main.m

```
%% Calibration
%  Calibration of extrinsic parameters.
load('calibration.mat');
global controller
if ~exist('tform')
    I = getFrame(2);
    tform = getTransform(I,25,cameraParameters);
end


%% Creation of work object
%  In order to create a work object, a piece of paper is used to
%  determine the natural work object according to the paper.
verifiedByHuman = 'no';
disp('Aim over the paper so that all edges are visible');

I = getFrame(2,1);

input('Start the rapid program, press enter when ready');
while ~strcmp(upper(verifiedByHuman(1)),'Y')
    I = getFrame(2,0);

    I = undistortImage(I,cameraParameters);

    [corners, center, paper] = getCornersPaper(I);
    figure,imshow(I)
    title('Coordinate system of defined workObject');


    imageCenter = [size(I,2),size(I,1)]./2;
    for j=1:length(corners)
        distance(j) = (corners(j,1).^2+corners(j,2).^2).^.5;
    end
    [value,index] = sort(distance);
    origo = corners(index(1),:);
    distanceToOrigo(1) = ((origo(1)-corners(index(2),1))^2...
    + (origo(2)-corners(index(2),2))^2).^.5;
    distanceToOrigo(2) = ((origo(1)-corners(index(3),1))^2...
    + (origo(2)-corners(index(3),2))^2).^.5;
    [valueO,indexO] =sort(distanceToOrigo);

    pointX = corners(index(indexO(1)+1),:);
    pointY = corners(index(indexO(2)+1),:);

    hold on
    plot(origo(1),origo(2),'bx','LineWidth',2);
    plot([origo(1),pointX(1)],...
    [origo(2),pointX(2)],'r','LineWidth',2);
    plot([origo(1),pointY(1)],...
    [origo(2),pointY(2)],'g','LineWidth',2);
    drawnow();
```

```
52      [dxOrigo, dyOrigo]   = getWorldDist(imageCenter,origo,tform);
53      [dxPointX,dyPointX]  = getWorldDist(imageCenter,pointX,tform);
54      [dxPointY,dyPointY]  = getWorldDist(imageCenter,center,tform);
55
56      %Angle between camera and paper, for indication only.
57      angle =atan2(dyPointX-dyOrigo,dxPointX-dxOrigo)*180/pi
58      controller = abb.connect('152.94.0.38');
59      currentState = abb.create(controller,'currentState');
60      nextState = abb.create(controller,'nextState');
61      displ1 = abb.create(controller,'displ1');
62      displ2 = abb.create(controller,'displ2');
63      displ3 = abb.create(controller,'displ3');
64      bord = abb.create(controller,'bord');
65      rotation = abb.create(controller,'rotation');
66      matlabPos = abb.create(controller,'matlabPos');
67      currentPos = abb.create(controller,'currentPos');
68      abb.setState(currentState,nextState,0);
69      distToTable = abb.get(currentPos,1).Z-abb.get(bord);
70      abb.set(displ1,[dxOrigo, dyOrigo,distToTable]);
71      abb.set(displ2,[dxPointX,dyPointX,distToTable]);
72      abb.set(displ3,[dxPointY,dyPointY,distToTable]);
73      abb.setState(currentState,nextState,1);
74      abb.setState(currentState,nextState,2);
75      abb.setState(currentState,nextState,99);
76      abb.setState(currentState,nextState,0);
77      abb.logoff(controller);
78      close all
79      figure,imshow(getFrame(2,0));
80      drawnow();
81      verifiedByHuman = ...
82      input('Is the camera in the center of the paper? [Y/N]','s');
83  end
84
85  %% Actual picking up part
86  again = ''
87  while strcmp(again,'') || length(again) > 0 &&...
88      strcmp(upper(again(1)),'Y')
89      close all
90      abb.logon(controller);
91      abb.setState(currentState,nextState,99);
92      I = getFrame(2);
93
94      I = undistortImage(I,cameraParameters);
95      color_markers = [
96
97        128.5905  122.3360; %background
98        164.3418  143.7367; %red
99        133.7915  181.5070; %yellow
100       113.3183  115.9544; %green
101       138.1641   90.3943; %blues
102
103     ];
104     names = {'Background','Red','Yellow','Green','Blue'};
105     [corners,center,paper] = getCornersPaper(I);
106
107     imageCenter = [size(I,2),size(I,1)]./2;
108     clear distance;
```

```
109     for j=1:length(corners)
110         distance(j) = (corners(j,1).^2+corners(j,2).^2).^.5;
111     end
112     [value,index] = sort(distance);
113     origo = corners(index(1),:);
114 
115     [images,combinedImage] = getSegmentedImage(paper,color_markers) ;
116     clear distance;
117     col = getCornersFromImages(images,names);
118     for colorCount = 1:length(col)
119         for objectCount = 1:length(col(colorCount).object)
120                 X = col(colorCount).object(objectCount).corners(:,1);
121                 Y = col(colorCount).object(objectCount).corners(:,2);
122                 center =col(colorCount).object(objectCount).center;
123                 x1 = X(1);
124                 y1 = Y(1);
125                 distance = ((x1 - X).^2 + (y1 -Y).^2).^.5;
126                 if length(distance) == 4
127                     [value,index] = sort(distance);
128                     %     x1--------------x2
129                     %     |               |
130                     %     p1      c       p2
131                     %     |               |
132                     %     x3--------------x4
133                     x2 = X(index(3));
134                     y2 = Y(index(3));
135                     x3 = X(index(2));
136                     y3 = Y(index(2));
137                     x4 = X(index(4));
138                     y4 = Y(index(4));
139 
140                     p1 = [(x1+x3)/2 (y1+y3)/2];
141                     p2 = [(x2+x4)/2 (y2+y4)/2];
142                     angle =atan2(p2(1)-p1(1),p2(2)-p1(2))*180/pi;
143                     if angle > 90
144                         angle = angle -180;
145                     end
146                     col(colorCount).object(objectCount).angle =...
147                    angle;
148                 end
149 
150         end
151     end
152 
153 
154     abb.setState(currentState,nextState,7);
155     for colorCount = 1:length(col)
156             numBricks = length(col(colorCount).object);
157             if(numBricks > 0)
158              disp(['Picking ' int2str(numBricks) ' '...
159              col(colorCount).name ' bricks'])
160             end
161         for objectCount = 1:length(col(colorCount).object)
162             center = col(colorCount).object(objectCount).center;
163             angle = col(colorCount).object(objectCount).angle;
164             if angle < 0
165                 configuration = [-1, 0, 0, 0];
```

```
166             elseif angle > 0 && angle < 28
167                 configuration = [-1, 0, 1, 0];
168             else
169                 configuration = [-1, -1, 2, 0];
170             end
171 
172 
173             [dxKloss, dyKloss]   = getWorldDist(origo,center,tform);
174 
175             abb.setState(currentState,nextState,0);
176             abb.set(matlabPos,[[dxKloss,dyKloss,-200],[1,0,0,0],...
177             configuration,[9E9,9E9,9E9,9E9,9E9,9E9]]);
178             abb.set(rotation,angle);
179             abb.setState(currentState,nextState,3);
180             abb.setState(currentState,nextState,5);
181             abb.setState(currentState,nextState,0);
182             abb.set(matlabPos,[[dxKloss,dyKloss,0],[1,0,0,0],...
183             configuration,[9E9,9E9,9E9,9E9,9E9,9E9]]);
184             abb.setState(currentState,nextState,3);
185             abb.setState(currentState,nextState,4);
186             abb.setState(currentState,nextState,9);
187             abb.setState(currentState,nextState,0);
188             abb.set(matlabPos,[[dxKloss,dyKloss,-200],...
189             [1,0,0,0],[-1,0,1,0],[9E9,9E9,9E9,9E9,9E9,9E9]]);
190             abb.set(rotation,0);
191             abb.setState(currentState,nextState,3);
192             abb.setState(currentState,nextState,5);
193             abb.setState(currentState,nextState,0);
194             if strcmp(col(colorCount).name,'Red')
195                 dumpingPos = [171.4,343.8];
196             elseif strcmp(col(colorCount).name,'Yellow')
197                 dumpingPos = [238,46];
198             elseif strcmp(col(colorCount).name,'Green')
199                 dumpingPos = [270.7,218.1];
200             elseif strcmp(col(colorCount).name,'Blue')
201                 dumpingPos = [50,343.8];
202             end
203             abb.set(rotation,0);
204             abb.set(matlabPos,[dumpingPos,-200,[1,0,0,0],...
205             [-1,0,1,0],[9E9,9E9,9E9,9E9,9E9,9E9]]);
206             abb.setState(currentState,nextState,3);
207             abb.setState(currentState,nextState,5);
208             abb.setState(currentState,nextState,0);
209             abb.set(matlabPos,[dumpingPos,-(5*objectCount),...
210             [1,0,0,0],[-1,0,1,0],[9E9,9E9,9E9,9E9,9E9,9E9]]);
211             abb.setState(currentState,nextState,4);
212             abb.setState(currentState,nextState,7);
213             abb.setState(currentState,nextState,0);
214             abb.set(matlabPos,[dumpingPos,-200,[1,0,0,0],...
215             [-1,0,1,0],[9E9,9E9,9E9,9E9,9E9,9E9]]);
216             abb.setState(currentState,nextState,3);
217             abb.setState(currentState,nextState,5);
218         end
219     end
220     again = input('Do want to run the program again [Y/N]?','s')
221 
222 end
```

B.2 getFrame

```
function frame = getFrame( winvideoID, enablePreview )
%GETFRAME Returns a single frame from winvideoID
%   This function returns a single frame from the
%   selected winvideo ID. If enablePreview is 1 the function
%   shows a preview from the camera before returning.
%   getFrame()
%   returns frame from the default camera without preview
%   getFrame(2,1)
%   returns a frame from the second camera with preview
    vid =  videoinput('winvideo',winvideoID);
    src = getselectedsource(vid);

    if(~exist('enablePreview'))
        enablePreview = 1;
    end

    pause('on');
    pause(3);
    if(enablePreview)
        start(vid);
        h = figure('Toolbar','none',...
               'Menubar', 'none',...
               'NumberTitle','Off',...
               'Name','Video Feed');

        vidRes = get(vid, 'VideoResolution');
        nBands = get(vid, 'NumberOfBands');
        hImage = image( zeros(vidRes(2), vidRes(1), nBands) );
        preview(vid,hImage);
        imageCenter = [vidRes(1),vidRes(2)]./2;
        hold on
        plot(imageCenter(1),imageCenter(2),'bx')
        plot(imageCenter(1)+100,imageCenter(2)+100,'bx')
        plot(imageCenter(1)+100,imageCenter(2)-100,'bx')
        plot(imageCenter(1)-100,imageCenter(2)-100,'bx')
        plot(imageCenter(1)-100,imageCenter(2)+100,'bx')
        plot(imageCenter(1)+imageCenter(1),imageCenter(2),'bx')
        plot(imageCenter(1)-imageCenter(1)+1,imageCenter(2),'bx')
        plot(imageCenter(1),imageCenter(2)*2,'bx')
        plot(imageCenter(1),imageCenter(2)*0+1,'bx')

        disp 'Press any key to capture frame...'
        pause;
        disp 'Captured!'
    end
    frame = getsnapshot(vid);
    if(enablePreview)
        closepreview(vid);
        close(h)
    end
    flushdata(vid);
    delete(vid);
end
```

B.3 getTransform

```
function tform = getTransform( I ,squareSize,cameraParameters)
%GETTRANSFORM returns a projective2d transform
%   This function returns a projective2d transform for usage in
%   transformPointsForward and transformPointsInverse.
%   The supplied image Ineeds to contain a checkerboard pattern.
%   tform = getTransform(I,25,cameraParamters);

    I = undistortImage(I, cameraParameters);
    [imageCorners, boardSize] = detectCheckerboardPoints(I);
    if(boardSize == [0 0])
        error('transformError:checkerboard',...
              'Error: no checkerboard pattern detected');
    end
    worldCorners = generateCheckerboardPoints(boardSize, squareSize);
    Kinv = inv(cameraParameters.IntrinsicMatrix);
    H = fitgeotrans(worldCorners, imageCorners, 'projective');
    H = H.T;
    h1 = H(1, :);
    h2 = H(2, :);
    h3 = H(3, :);
    lambda = 1 / norm(h1 * Kinv) - 19.15;
    r1 = lambda * h1 * Kinv;
    r2 = lambda * h2 * Kinv;
    r3 = cross(r1, r2);
    R = [r1; r2; r3];
    [U, ~, V] = svd(R);
    R = U * V';
    t = lambda * h3 * Kinv;
    T = [R(1, :); R(2, :); t] * cameraParameters.IntrinsicMatrix;
    tform = projective2d(T);
end
```

B.4 getWorldDist

```
function [dx,dy] = getWorldDist(pictureStart,pictureStop,tform)
%GETWORLDDIST returns the distance between two points in millimetres
%   The function uses a projective2d transform to calulate
%   the distance between two points in the picture and return
%   the result in millimetres. The result two scalars
%   dx is distance along the x-axis and
%   dy is distance along the y-axis.
%   [distanceX,distance] = getWorldDist(A,B,tform)
    [sta(1),sta(2)] = transformPointsInverse(tform,...
                      pictureStart(1),pictureStart(2));
    [sto(1),sto(2)] = transformPointsInverse(tform,...
                      pictureStart(1),pictureStop(2));
    dx =sign(pictureStop(2)-pictureStart(2))*norm(sta-sto);
    [sta(1),sta(2)] = transformPointsInverse(tform,...
                      pictureStart(1),pictureStart(2));
    [sto(1),sto(2)] = transformPointsInverse(tform,...
                       pictureStop(1),pictureStart(2));
    dy =sign(pictureStop(1)-pictureStart(1))*norm(sta-sto);

end
```

B.5 getColorMarkers

```
function colorMarkers = getColorMarkers( I,nColors,names )
%GETCOLORMARKERS Generates colorMarkers from user input
%   Returns colorMarkers for use with getSegmentedImage.
%   cm = getColorMarkers(I,4,{'Red','Green','Blue','Yellow'})
    h = fspecial('gaussian', [25 25], 4);
    cform = makecform('srgb2lab');
    lab_i = applycform(imfilter(I, h),cform);
    lab_i(:,:,1) = 0;
    if(exist('names'))
        if(length(names)~= nColors)
            clear names
        end
    end
    for count = 1:nColors
        if(exist('names'))
            disp(['Now selecting ' names{count} ])
        end
        sample_regions(:,:,count) = roipoly(I);
    end
    a = lab_i(:,:,2);
    b = lab_i(:,:,3);
    for count = 1:nColors
      colorMarkers(count,1) = mean2(a(sample_regions(:,:,count)));
      colorMarkers(count,2) = mean2(b(sample_regions(:,:,count)));
    end
end
```

B.6 getSegmentedImage

```
function [images,combinedImage] = getSegmentedImage(I,colorMarkers,h)
%GETSEGMENTEDIMAGE Segments image into segments basec on colorMarkers
%   The segments are binary images and their order correspond to the
%   order of the color markers. A set of combined images are returned
%   in combinedImage
%   The variable colorMarkers are generated from getColorMarkers.
%   [im,cm] = getSegmentedImage(I,colorMarkers);
%   uses the standard 25x25, sigma = 5 gaussian filter
%   [im,cm] = getSegmentedImage(I,colorMarkers,1);
%   uses no filter
%   [im,cm] = getSegmentedImage(I,colorMarkers,h);
%   uses the kernel h
    if(~exist('h'))
        h = fspecial('gaussian', [25 25], 5);
    end
    I = imfilter(I,h);


    cform = makecform('srgb2lab');
    lab_I = applycform(I,cform);

    a = lab_I(:,:,2);
    b = lab_I(:,:,3);


    nColors = length(colorMarkers);
    a = double(a);
    b = double(b);
    distance = repmat(0,[size(a), nColors]);

    for count = 1:nColors
      distance(:,:,count) = ((a - colorMarkers(count,1)).^2...
                             + (b - colorMarkers(count,2)).^2).^.5;
    end

    [value, label] = min(distance,[],3);

    images = repmat(logical(false),[size(I,1),size(I,2), nColors]);
    combinedImage = repmat(uint8(0),[size(I,1),size(I,2)]);


    for count = 1:nColors
        images(:,:,count) = label == count;
        images(:,:,count) = imclearborder(images(:,:,count),4);
        images(:,:,count) = bwareaopen(images(:,:,count),...
                                       floor(numel(I)/3000));
        images(:,:,count) = bwfill(images(:,:,count),'holes');
        combinedImage = combinedImage+...
                        uint8(images(:,:,count)*count);
    end


end
```

B.7 getCorners

```
function [X,Y,center] = getCorners(I)
%GETCORNERS Returns the coordinates of the corners in the image I
%   The image is assumed to be binary and only have one object in it
%   If something different from four corners is found, a plot of the
%   angle vs distance is shown.
%   The corners need to be pi/4 apart from eachother to be detected.
%   [X,Y,c] = getCorners(I);
    edges = edge(I,'canny');
    [y,x] = find(edges==1);
    center = [round(mean(x)) round(mean(y))];

    [th,rho] = cart2pol(x-center(1),y-center(2));
    [k,ind] = sort(th);
    th = th(ind);
    rho = smooth(rho(ind));
    [val,ind] = findpeaks(rho,'MINPEAKHEIGHT',...
                max(rho)*0.9,'MINPEAKDISTANCE',round(length(th)/8));


    if length(ind) == 4
        disp 'Found four corners'
    elseif length(ind) == 5
        disp 'Found five corners, retrying...'
        rho = circshift(rho, round(length(th)/4));
        [val,ind] = findpeaks(rho,'MINPEAKHEIGHT',...
                max(rho)*0.9,'MINPEAKDISTANCE',round(length(th)/8));
        ind = (ind-round(length(th)/4));
        ind(find(ind<0))=ind(find(ind<0)) +length(th);

        rho = circshift(rho, -round(length(th)/4));
    else
        disp 'Rectangular shape not detected'
        figure, plot(th,rho)
        hold on
        plot(th(ind),rho(ind),'o');

    end

    [X,Y] = pol2cart(th(ind),val);

    X = X+center(1);
    Y = Y+center(2);
end
```

B.8 getCornersPaper

```
function [corners, center, paper] = getCornersPaper(I)
%GETCORNERSPAPER gets all corners of a piece of paper.
%   Returns the corners as a vector, the center
%   and the contents on the found piece of paper.
%   [corners,center,paper] = getCorners(I);
    image = rgb2gray(I);

    %the actual threshold is somewhat primitive
    %but has to worked fine
    bw = image > 120;
    bw = bwfill(bw,'holes');

    maxSize = floor(size(image,1)*size(image,2)*0.1);
    bw = bwareaopen(bw,maxSize);
    image(:,:,1) = I(:,:,1).*uint8(bw);
    image(:,:,2) = I(:,:,2).*uint8(bw);
    image(:,:,3) = I(:,:,3).*uint8(bw);

    [X,Y,c] = getCorners(bw);
    center = c;
    corners = [X,Y];
    paper = image;
    if size(X) < 4
        imshow(bw);
        error('Corners not found');
    end

end
```

B.9 getCornersFromImages

```
function col = getCornersFromImages( images,names )
%GETCORNERSFROMIMAGES Uses finds corners in a set of images
%   Returns a structure containing names, corners and centers
%   Uses images from getSegmentedImages
%col = getCornersFromImages(images,{'Red','Green','Blue','Yellow'});
    if(~exist('names'))
        for k=1:size(images,3)
            names{k} = 'Undefined';
        end
    end
    for count=1:size(images,3)
        boundaries = bwboundaries(images(:,:,count));
        col(count).name = names{count};
        objCount = 1;
        for k=1:length(boundaries)
            b = boundaries{k};
                BW_segment = repmat(logical(false),...
                 [(max(b(:,1))- min(b(:,1)))+5,...
                 (max(b(:,2))- min(b(:,2)))+5 ]);
                BW_segment(3:end-2,3:end-2) = images(...
                  min(b(:,1)):max(b(:,1)),...
                  min(b(:,2)):max(b(:,2)),count);
                [X,Y,c] = getCorners(BW_segment);
                X = X + min(b(:,2));
                Y = Y + min(b(:,1));
                c(1) = c(1) + min(b(:,2));
                c(2) = c(2) + min(b(:,1));
                if(length(X) == 4)
                    col(count).object(objCount).corners = [X Y];
                    col(count).object(objCount).center = c;
                    objCount = objCount + 1;
                end
        end
    end

end
```

B.10 abb.connect

```
function ctrl = connect( IP, number )
%CONNECT connects to an ABB controller
%    Returns the controller object at the specified IP address
%    the global variable controller is also set
%    ctrl = abb.connect('localhost',1)
%    connects to the first controller on localhost.
%    ctrl = abb.connect('localhost',2)
%    connects to the second controller on localhost.
%    ctrl = abb.connect(11.22.33.44)
%    connects to the controller on 11.22.33.44

    global controller;
    if strcmp(IP,'localhost')
        IP = '127.0.0.1';
    end
    try
        NET.addAssembly('ABB.Robotics');
        NET.addAssembly('ABB.Robotics.Controllers');
        scan =  ABB.Robotics.Controllers.Discovery.NetworkScanner();
        scan.Scan();
        cic = scan.Controllers();
        if cic.Count == 0
            pause(2)
            cic = scan.Controllers();
        end
        controllers = scan.GetControllers();

        validControllers = [];
        for i=1:controllers.Length
            if controllers(i).IPAddress.ToString == IP
                validControllers = [validControllers i];
            end
        end
        if length(validControllers) == 0
            error('ABB:ConnectError',...
                  'No controller found on specified IP');
        elseif length(validControllers) > 1
           if exist('number') & isa(number,'double')
               controller = controllers(validControllers(number));
           else
               error('ABB:ConnectError',...
                   ['Several controllers found on specified IP,',...
' please use connect(IP,index) where index',...
' is the specified controller. ']);

           end
        elseif length(validControllers) == 1
            controller = controllers(validControllers);
        end
        ctrl = ABB.Robotics.Controllers.Controller(controller);
        ctrl.Logon( ABB.Robotics.Controllers.UserInfo.DefaultUser );
        master = ABB.Robotics.Controllers.Mastership.Request(...
                 ctrl.Rapid);
        name = controller.Name.char;
```

```
55          realIP = controller.IPAddress.ToString.char;
56          user = ctrl.CurrentUser.Name.char;
57          masterString = '';
58          if(master.IsMaster == 1)
59              masterString = ' (master)';
60          end
61          disp(['Connected to ' name '...
62                ' at ' realIP 'as ' user masterString]);
63      catch me
64          error('ABB:LoginError' ,...
65              ['Connection failed:\n' ...
66              me.message  '\nLine: ' int2str(me.stack.line)]);
67      end
68      controller = ctrl;
69
70  end
```

B.11 abb.disconnect

```
function disconnect( ctrl )
%DISCONNECT Disconnect from the specified controller
%   Distonnects and destroys the controller object
%   abb.disconnet(ctrl)
    name = ctrl.RobotWare.Name.char;
    realIP = ctrl.IPAddress.ToString.char;
    user = ctrl.CurrentUser.Name.char;
    disp(['Disconnected from ' name ' at ' realIP ' as ' user])
    ctrl.Logoff();
    ctrl.Dispose();
    ctrl.delete();
end
```

B.12 abb.logon

```
function logon( ctrl )
global controller;
%LOGON Logs on to the specified controller
%   In case of a disconnect or communication error
%   this function logs back onto an existing controller
%   object.
%   abb.logon(ctrl)
%   logs back on to the controller ctrl
%   abb.logon()
%   logs back onto the global variable controller
%   this is usually the last controller logged on to
    if(~exist('ctrl') && exist('controller'))
        ctrl = controller;
    end
    if (ctrl.IsMaster ~= 1)
        ctrl.Logon( ABB.Robotics.Controllers.UserInfo.DefaultUser );
        master = ABB.Robotics.Controllers.Mastership.Request(...
                 ctrl.Rapid);
        name = ctrl.RobotWare.Name.char;
        realIP = ctrl.IPAddress.ToString.char;
        user = ctrl.CurrentUser.Name.char;
        masterString = '';
        if(master.IsMaster == 1)
            masterString = ' (master)';
        end
        disp(['Logged on to ' name '...
             at ' realIP ' as ' user masterString]);
    end
end
```

B.13 abb.logoff

```
function logoff( ctrl )
%LOGOFF Logs off a controller
%   Similarily to disconnect, this logs off a controller
%   however, the object is not destroyed
%   logoff(ctrl);
    name = ctrl.RobotWare.Name.char;
    realIP = ctrl.IPAddress.ToString.char;
    user = ctrl.CurrentUser.Name.char;
    disp(['Logged off from ' name ' at ' realIP ' as ' user])
    ctrl.Logoff();
end
```

B.14 abb.create

```
function rapidReference = create( ctrl, variable, program, module )
%CREATE Creates a new reference to a RAPID variable
%   program and module will default to T_ROB1 and Module1
%   abb.create(ctrl, variable, program, module)
%   abb.create(ctrl, variable)
if ~exist('program')
    program = 'T_ROB1';
end

if ~exist('module')
    module = 'Module1';
end

NETarray = NET.createArray('System.String',3);
NETarray.Set(0, program);
NETarray.Set(1, module);
NETarray.Set(2, variable);
rapidData = ctrl.Rapid.GetRapidData( NETarray );
rapidReference = rapidData;

end
```

B.15 abb.get

```
function value = get( rapidData, returnStructure)
%GET reads value from RAPID
%   By setting returnStructore to 1 a structure is returned insted
%   rapidData is a reference created with abb.create
%   abb.get(status)
%   returns the value of status
%   abb.get(status,1)
%   returns the value of status as a structure

    if(~exist('returnStructure'))
        returnStructure = 0;
    end
    argument1 = inputname(1);
    if strcmp(argument1,'')
       argument1 = 'rapidData';
    end
    if(~isa(rapidData,...
       'ABB.Robotics.Controllers.RapidDomain.RapidData'))
        error('ABB:RapidError',...
          [argument1...
          ' is not a valid ABB.Robotics.',...
          'Controllers.RapidDomain.RapidData']);
    end

    if(strcmp(rapidData.RapidType.char,'num') )
        value = rapidData.Value.Value;
    elseif(strcmp(rapidData.RapidType.char,'pos') &&...
           returnStructure)
        value.X = rapidData.Value.X;
        value.Y = rapidData.Value.Y;
        value.Z = rapidData.Value.Z;
    elseif(strcmp(rapidData.RapidType.char,'pos') &&...
           ~returnStructure)
        value(1) = rapidData.Value.X;
        value(2) = rapidData.Value.Y;
        value(3) = rapidData.Value.Z;
    elseif(strcmp(rapidData.RapidType.char,'pose') &&...
           returnStructure)
        value.X = rapidData.Value.Trans.X;
        value.Y = rapidData.Value.Trans.Y;
        value.Z = rapidData.Value.Trans.Z;
        value.Q1 = rapidData.Value.Rot.Q1;
        value.Q2 = rapidData.Value.Rot.Q2;
        value.Q3 = rapidData.Value.Rot.Q3;
        value.Q4 = rapidData.Value.Rot.Q4;
    elseif(strcmp(rapidData.RapidType.char,'pose') &&...
           ~returnStructure)
        value(1) = rapidData.Value.Trans.X;
        value(2) = rapidData.Value.Trans.Y;
        value(3) = rapidData.Value.Trans.Z;
        value(4) = rapidData.Value.Rot.Q1;
        value(5) = rapidData.Value.Rot.Q2;
        value(6) = rapidData.Value.Rot.Q3;
        value(7) = rapidData.Value.Rot.Q4;
```

```
55      elseif(strcmp(rapidData.RapidType.char,'robtarget') &&...
56             returnStructure)
57          value.X = rapidData.Value.Trans.X;
58          value.Y = rapidData.Value.Trans.Y;
59          value.Z = rapidData.Value.Trans.Z;
60          value.Q1 = rapidData.Value.Rot.Q1;
61          value.Q2 = rapidData.Value.Rot.Q2;
62          value.Q3 = rapidData.Value.Rot.Q3;
63          value.Q4 = rapidData.Value.Rot.Q4;
64          value.Cf1 = rapidData.Value.Robconf.Cf1;
65          value.Cf4 = rapidData.Value.Robconf.Cf4;
66          value.Cf6 = rapidData.Value.Robconf.Cf6;
67          value.Cfx = rapidData.Value.Robconf.Cfx;
68          value.Eax_a = rapidData.Value.Extax.Eax_a;
69          value.Eax_b = rapidData.Value.Extax.Eax_b;
70          value.Eax_c = rapidData.Value.Extax.Eax_c;
71          value.Eax_d = rapidData.Value.Extax.Eax_d;
72          value.Eax_e = rapidData.Value.Extax.Eax_e;
73          value.Eax_f = rapidData.Value.Extax.Eax_f;
74      elseif(strcmp(rapidData.RapidType.char,'robtarget') &&...
75              ~returnStructure)
76          value(1) = rapidData.Value.Trans.X;
77          value(2) = rapidData.Value.Trans.Y;
78          value(3) = rapidData.Value.Trans.Z;
79          value(4) = rapidData.Value.Rot.Q1;
80          value(5) = rapidData.Value.Rot.Q2;
81          value(6) = rapidData.Value.Rot.Q3;
82          value(7) = rapidData.Value.Rot.Q4;
83          value(8) = rapidData.Value.Robconf.Cf1;
84          value(9) = rapidData.Value.Robconf.Cf4;
85          value(10) = rapidData.Value.Robconf.Cf6;
86          value(11) = rapidData.Value.Robconf.Cfx;
87          value(12) = rapidData.Value.Extax.Eax_a;
88          value(13) = rapidData.Value.Extax.Eax_b;
89          value(14) = rapidData.Value.Extax.Eax_c;
90          value(15) = rapidData.Value.Extax.Eax_d;
91          value(16) = rapidData.Value.Extax.Eax_e;
92          value(17) = rapidData.Value.Extax.Eax_f;
93      end
94  end
```

B.16 abb.set

```
function set( rapidData, value )
%SET Sets a rapid variable to value
%    This sets the value of a rapidData reference.
%    Structures can be used to set this variable as well.
%    abb.set(rapidNum, 3)
%    abb.set(rapidRobtarget, [[x,y,z], [q1,q2,q3,q4],...
%   [cf1,cf4,cf6,cfx],[9e9,9e9,9e9,9e9,9e9,9e9]])
%    abb.set(rapidRobtarget, [[x,y,z], [q1,q2,q3,q4]]
%    abb.set(rapidRobtarget, [[x,y,z]]
%    abb.set(rapidPos, [[x,y,z]]
%    abb.set(rapidPose, [[x,y,z], [q1,q2,q3,q4]]
%    abb.set(rapidPose, [[x,y,z]]

    global controller;
    %Error handling
    timeout = 5;
    argument1 = inputname(1);
    argument2 = inputname(2);
    if strcmp(argument1,'')
       argument1 = 'rapidData';
    end
    if strcmp(argument2,'')
       argument2 = 'value';
    end
    if(~isa(rapidData,...
       'ABB.Robotics.Controllers.RapidDomain.RapidData'))
        error('ABB:RapidError',...
[argument1 '...
' is not a valid ABB.Robotics.Controllers.RapidDomain.RapidData']);
    end
    if(strcmp(rapidData.Symbol.Type.char, 'Constant'))
        error('ABB:RapidError',...
              [argument1 ' is a constant.']);

    end

    communicationFail = 1;
    while(communicationFail)
        try


        if(~isempty(whos('global','controller')))

            if controller.IsMaster == 0
                abb.logon(controller)
            end
        else
                error('ABB:RapidError',...
                [argument1 ' controller is not initialized.']);
        end


            if(exist('value') && isa(value,'struct'))
                %for when value is a structure
```

```
55                    %Type handling for num type
56                  if(strcmp(rapidData.RapidType.char,'num'))
57                      if(length(fieldnames(value))==1)
58                          rapidData.Value.Value =  value.Value;
59                      else
60                         error('ABB:RapidError',...
61                            [argument2 ' is not a valid '...
62                            rapidData.RapidType.char '.']);
63                      end
64
65                  end
66
67                  %Type handling for robtarget
68                  if(strcmp(rapidData.RapidType.char,'robtarget'))
69                      if(length(fieldnames(value))==3)
70                          rapidData.Value.Trans.X = value.X;
71                          rapidData.Value.Trans.Y = value.Y;
72                          rapidData.Value.Trans.Z = value.Z;
73                      elseif(length(fieldnames(value))==7)
74                          rapidData.Value.Trans.X = value.X;
75                          rapidData.Value.Trans.Y = value.Y;
76                          rapidData.Value.Trans.Z = value.Z;
77                          rapidData.Value.Rot.Q1 = value.Q1;
78                          rapidData.Value.Rot.Q2 = value.Q2;
79                          rapidData.Value.Rot.Q3 = value.Q3;
80                          rapidData.Value.Rot.Q4 = value.Q4;
81                      elseif(length(fieldnames(value))==17)
82                          rapidData.Value.Trans.X = value.X;
83                          rapidData.Value.Trans.Y = value.Y;
84                          rapidData.Value.Trans.Z = value.Z;
85                          rapidData.Value.Rot.Q1 = value.Q1;
86                          rapidData.Value.Rot.Q2 = value.Q2;
87                          rapidData.Value.Rot.Q3 = value.Q3;
88                          rapidData.Value.Rot.Q4 = value.Q4;
89                          rapidData.Value.Robconf.Cf1 = value.Cf1;
90                          rapidData.Value.Robconf.Cf4 = value.Cf4;
91                          rapidData.Value.Robconf.Cf6 = value.Cf6;
92                          rapidData.Value.Robconf.Cfx = value.Cfx;
93                          rapidData.Value.Extax.Eax_a = value.Eax_a;
94                          rapidData.Value.Extax.Eax_b = value.Eax_b;
95                          rapidData.Value.Extax.Eax_c = value.Eax_c;
96                          rapidData.Value.Extax.Eax_d = value.Eax_d;
97                          rapidData.Value.Extax.Eax_e = value.Eax_e;
98                          rapidData.Value.Extax.Eax_f = value.Eax_f;
99                      else
100                        error('ABB:RapidError',...
101                           [argument2 ' is not a valid '...
102                           rapidData.RapidType.char '.']);
103                     end
104
105                 end
106
107                 %Type handling for pos
108                 if(strcmp(rapidData.RapidType.char,'pos'))
109                     if(isequal(size(value),[1 3]) &&...
110                     isa(value,'numeric'))
111
```

```
112                         rapidData.Value.X = value.X;
113                         rapidData.Value.Y = value.Y;
114                         rapidData.Value.Z = value.Z;
115                     else
116                        error('ABB:RapidError',...
117                           [argument2 ' is not a valid '...
118                           rapidData.RapidType.char '.']);
119                     end
120                 end
121 
122                 %Type handling for pose
123                 if(strcmp(rapidData.RapidType.char,'pose'))
124                     if(isequal(size(value),[1 3]) &&...
125                     isa(value,'numeric'))
126                         rapidData.Value.Trans.X = value.X;
127                         rapidData.Value.Trans.Y = value.Y;
128                         rapidData.Value.Trans.Z = value.Z;
129                     elseif(isequal(size(value),[1 7]) &&...
130                     isa(value,'numeric'))
131                         rapidData.Value.Trans.X = value.X;
132                         rapidData.Value.Trans.Y = value.Y;
133                         rapidData.Value.Trans.Z = value.Z;
134                         rapidData.Value.Rot.Q1 = value.Q1;
135                         rapidData.Value.Rot.Q2 = value.Q2;
136                         rapidData.Value.Rot.Q3 = value.Q3;
137                         rapidData.Value.Rot.Q4 = value.Q4;
138                     else
139                        error('ABB:RapidError',...
140                           [argument2 ' is not a valid '...
141                           rapidData.RapidType.char '.']);
142                     end
143                 end
144                 t1 = clock;
145                 while(etime(clock,t1)<timeout &&...
146                 ~abb.isequal(value,abb.get(rapidData)))
147                     %wait until timeout or equality
148                 end
149                 etime(clock,t1)
150                 if(~abb.isequal(value,abb.get(rapidData)) &&...
151                    etime(clock,t1)>timeout)
152                     error('ABB:RapidError',...
153                           ['Could not set value.',...
154                            ' (timed out after '...
155                            num2str(timeout) ' seconds)']);
156                 end
157             elseif(exist('value') && ~isa(value,'struct'))
158                 %Type handling for num type
159                 if(strcmp(rapidData.RapidType.char,'num'))
160                     if(isequal(size(value),[1 1]) &&...
161                        isa(value,'numeric'))
162                         rapidData.Value.Value = value;
163                     else
164                        error('ABB:RapidError',...
165                           [argument2 ' is not a valid '...
166                           rapidData.RapidType.char '.']);
167                     end
168 
```

```
169                 end
170
171                 %Type handling for robtarget
172                 if(strcmp(rapidData.RapidType.char,'robtarget'))
173                     if(isequal(size(value),[1 3]) &&...
174                     isa(value,'numeric'))
175                         rapidData.Value.Trans.X = value(1);
176                         rapidData.Value.Trans.Y = value(2);
177                         rapidData.Value.Trans.Z = value(3);
178                     elseif(isequal(size(value),[1 7]) &&...
179                     isa(value,'numeric'))
180                         rapidData.Value.Trans.X = value(1);
181                         rapidData.Value.Trans.Y = value(2);
182                         rapidData.Value.Trans.Z = value(3);
183                         rapidData.Value.Rot.Q1 = value(4);
184                         rapidData.Value.Rot.Q2 = value(5);
185                         rapidData.Value.Rot.Q3 = value(6);
186                         rapidData.Value.Rot.Q4 = value(7);
187                     elseif(isequal(size(value),[1 17]) &&...
188                     isa(value,'numeric'))
189                         rapidData.Value.Trans.X = value(1);
190                         rapidData.Value.Trans.Y = value(2);
191                         rapidData.Value.Trans.Z = value(3);
192                         rapidData.Value.Rot.Q1 = value(4);
193                         rapidData.Value.Rot.Q2 = value(5);
194                         rapidData.Value.Rot.Q3 = value(6);
195                         rapidData.Value.Rot.Q4 = value(7);
196                         rapidData.Value.Robconf.Cf1 = value(8);
197                         rapidData.Value.Robconf.Cf4 = value(9);
198                         rapidData.Value.Robconf.Cf6 = value(10);
199                         rapidData.Value.Robconf.Cfx = value(11);
200                         rapidData.Value.Extax.Eax_a = value(12);
201                         rapidData.Value.Extax.Eax_b = value(13);
202                         rapidData.Value.Extax.Eax_c = value(14);
203                         rapidData.Value.Extax.Eax_d = value(15);
204                         rapidData.Value.Extax.Eax_e = value(16);
205                         rapidData.Value.Extax.Eax_f = value(17);
206                     else
207                        error('ABB:RapidError',...
208                           [argument2 ' is not a valid '...
209                            rapidData.RapidType.char '.']);
210                     end
211
212                 end
213
214                 %Type handling for pos
215                 if(strcmp(rapidData.RapidType.char,'pos'))
216                     if(isequal(size(value),[1 3]) &&...
217                     isa(value,'numeric'))
218                         rapidData.Value.X = value(1);
219                         rapidData.Value.Y = value(2);
220                         rapidData.Value.Z = value(3);
221                     else
222                        error('ABB:RapidError',...
223                           [argument2 ' is not a valid '...
224                           rapidData.RapidType.char '.']);
225                     end
```

```
226                 end
227
228                 %Type handling for pose
229                 if(strcmp(rapidData.RapidType.char,'pose'))
230                     if(isequal(size(value),[1 3]) &&...
231                     isa(value,'numeric'))
232                         rapidData.Value.Trans.X = value(1);
233                         rapidData.Value.Trans.Y = value(2);
234                         rapidData.Value.Trans.Z = value(3);
235                     elseif(isequal(size(value),[1 7]) &&...
236                     isa(value,'numeric'))
237                         rapidData.Value.Trans.X = value(1);
238                         rapidData.Value.Trans.Y = value(2);
239                         rapidData.Value.Trans.Z = value(3);
240                         rapidData.Value.Rot.Q1 = value(4);
241                         rapidData.Value.Rot.Q2 = value(5);
242                         rapidData.Value.Rot.Q3 = value(6);
243                         rapidData.Value.Rot.Q4 = value(7);
244                     else
245                        error('ABB:RapidError',...
246                           [argument2 ' is not a valid '...
247                           rapidData.RapidType.char '.']);
248                     end
249                 end
250                 t1 = clock;
251
252
253                 while(etime(clock,t1)<timeout &&...
254                      ~abb.isequal(value,abb.get(rapidData)))
255                     %wait until timeout or equality
256                 end
257                 if(~abb.isequal(value,abb.get(rapidData)) &&...
258                     etime(clock,t1)>timeout)
259                     error('ABB:RapidError',...
260                           ['Could not set value.',...
261                            ' (timed out after '...
262                            num2str(timeout) ' seconds)']);
263                     end
264                 end
265                 communicationFail = 0;
266         catch
267             communicationFail = 1;
268         end
269     end
270
271
272
273 end
```

B.17 abb.isequal

```
function result = isequal( a,b )
%COMPARISON Summary of this function goes here
%   This function is used to compare two rapidData objects
%   isequal(a,b)
    result = 1;
    if(isa(a,'struct') && isa(b,'struct'))
        %check to see if they contain the same values
        aFn = fieldnames(a);
        bFn = fieldnames(b);
        if(isequal(size(aFn),size(bFn)))
            for k=1:size(aFn,1)

                    if(max(abs(a.(aFn{k})),...
                       abs(b.(bFn{k}))) >= 1000)
                        precision = 0.01;
                    elseif(max(abs(a.(aFn{k})),...
                           abs(b.(bFn{k}))) >= 100)
                        precision = 0.001;
                    elseif(max(abs(a.(aFn{k})),...
                           abs(b.(bFn{k}))) >= 10)
                        precision = 0.0001;
                    else
                        precision = 0.00001;
                    end
                    if(min(sign(a.(aFn{k})),sign(b.(bFn{k})))<0)
                        precision = precision*10;
                    end
                if(~strcmp(aFn{k},bFn{k}))
                    result = 0;
                elseif(abs(a.(aFn{k}) - b.(aFn{k})) > precision)
                    result = 0;
                end
            end
        else
            result = 0;
        end
    elseif(~isa(a,'struct') && ~isa(b,'struct') &&...
            isequal(size(a),size(b)) && numel(a) > 1)
        for k=1:numel(a)
            if(max(abs(a(k)),abs(b(k))) >= 1000)
                precision = 0.01;
            elseif(max(abs(a(k)),abs(b(k))) >= 100)
                precision = 0.001;
            elseif(max(abs(a(k)),abs(b(k))) >= 10)
                precision = 0.0001;
            else
                precision = 0.00001;
            end
            if(min(sign(a(k)),sign(b(k)))<0)
                precision = precision*10;
            end
            if(abs(single(a(k)) - single(b(k))) > precision)
                result = 0;
            end
```

```
55          end
56      elseif(~isa(a,'struct') && ~isa(b,'struct') &&...
57              isequal(size(a),size(b)) && numel(a) == 1)
58          if(max(abs(a),abs(b)) >= 1000)
59              precision = 0.01;
60          elseif(max(abs(a),abs(b)) >= 100)
61              precision = 0.001;
62          elseif(max(abs(a),abs(b)) >= 10)
63              precision = 0.0001;
64          else
65              precision = 0.00001;
66          end
67          if(min(sign(a),sign(b))<0)
68              precision = precision*10;
69          end
70          if (abs(a-b) > precision)
71              result = 0;
72          end
73
74      else
75                  error('Math:roundingError',...
76                    ['Incompatible types']);
77
78      end
79
80
81  end
```

B.18 abb.setState

```
function setState( currentState, wantedState, value)
%SETSTATE Simplified function to change state in RAPID
%   Roughly the same as set, but times out if currentState does not
%   equal value after 600 seconds.
%   setState(cs,ws,5);
    timeout = 600;
    if(abb.get(currentState) == abb.get(wantedState) &&...
       abb.get(wantedState) == value )
        %state is already set
    else
        abb.set(wantedState,value);
        t1 = clock;
        while(etime(clock,t1)<timeout &&...
        ~isequaln(value,abb.get(currentState)))
            %wait until timeout or equality
        end
        if(~isequaln(value,abb.get(currentState)) &&...
            etime(clock,t1)>timeout)
            error('ABB:RapidError',...
                  ['Could not set value. (timed out after '...
                  num2str(timeout) ' seconds)']);
        end

    end

end
```

C Rapid Kode

```
MODULE Module1

        const num bord :=-20;
        VAR robtarget nullPos :=[[105,148.5,0]
        ,[1,0,0,0],[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];
        VAR robtarget zero :=[[105,148.5,-400],[1,0,0,0]
        ,[0,-1,0,0],[9E9,9E9,9E9,9E9,9E9,9E9]];


        PERS tooldata tGripper:=[TRUE,[[-12.75,0,154.23],
        [1,0,0,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];
        PERS tooldata tGripperGrippingPos:=[TRUE,[[-12.75,0,154.23]
        ,[0.965926,0,-0.258819,0]],[1,[0,0,1],[1,0,0,0],0,0,0]];
        PERS tooldata tGripperCam:=[TRUE,[[53.69,0,41.19],
        [0.997969,0.00541994,0.0171534,-0.0611147]],
        [1,[0,0,1],[1,0,0,0],0,0,0]];


        VAR robtarget testpos;
        VAR robtarget p1;
        VAR robtarget p2;
        VAR robtarget p3;
        VAR robtarget currentPos;
        VAR robtarget matlabPos;
        VAR pos displ1;
        VAR pos displ2;
        VAR pos displ3;

        PERS wobjdata workGenerert:= [FALSE, TRUE, "",
        [[454.226, -390.902, -29.997],
        [2.92894E-06, -0.835016, 0.550226, 3.28752E-05]],
        [[0, 0, 0],[1, 0, 0, 0]]];
        VAR num currentState := 0;
        VAR num nextState := 0;
        VAR num rotation := 0;
        VAR pose frame;


    PROC main()
        tGripperCam.tframe.rot := OrientZYX(-7,2,0.5);
        tGripperGrippingPos.tframe.rot := OrientZYX(0,-30,0);
        stateMachine;
    ENDPROC

    PROC stateMachine()
           currentPos := CRobT(\Tool:=tGripperCam\WObj:=wobj0);
           testpos := CRobT(\Tool:=tGripper\WObj:=workGenerert);
        IF currentState = 0 THEN
            !idle
        ELSEIF currentState = 1 THEN
            p1 := RelTool (currentPos,
           displ1.x,  displ1.y , displ1.z);
            p2 := RelTool (currentPos,
```

```
54              displ2.x , displ2.y , displ2.z);
55              p3 := RelTool (currentPos,
56              displ3.x,  displ3.y , displ3.z);
57              frame := DefFrame(p1,p2,p3);
58              workGenerert.uframe := frame;
59
60          ELSEIF currentState = 3 THEN
61              matlabPos.rot := OrientZYX(rotation,0,0);
62
63          ELSEIF currentState = 4 THEN
64              !Go to matlabPos with the gripper
65              MoveL matlabPos,v7000,fine,
66              tGripperGrippingPos\WObj:=workGenerert;
67
68          ELSEIF currentState = 5 THEN
69              !Go to matlabPos with the gripper in gripping pos
70              MoveJ matlabPos,v7000,z200,
71              tGripperGrippingPos\WObj:=workGenerert;
72
73          ELSEIF currentState = 6 THEN
74              !Go to matlabPos with the camera
75              MoveL matlabPos,v7000,fine,
76              tGripperCam\WObj:=workGenerert;
77
78          ELSEIF currentState = 7 THEN
79              !Close the gripper
80              SetDO doGripClose, 1;
81              SetDO doGripOpen, 1;
82              WaitTime 0.2;
83
84          ELSEIF currentState = 9 THEN
85              !Open the gripper
86              SetDO doGripOpen, 0;
87              SetDO doGripClose, 1;
88              WaitTime .2;
89
90          ELSEIF currentState = 98 THEN
91              MoveL zero,v7000,fine,
92              tGripper\WObj:=workGenerert;
93
94          ELSEIF currentState = 99 THEN
95              MoveL zero,v7000,fine,
96              tGripperCam\WObj:=workGenerert;
97
98          ENDIF
99          currentState := nextState;
100
101     endproc
102
103 ENDMODULE
```