Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Bachelor i ingeniørfag / Automatisering og elektronikkdesign	Vårsemesteret 2021 Åpen
Forfatter(e): Mari Amdalsrød Hognestad, Sander André Søndeland	
Fagansvarlig: Karl Skretting Veileder(e): Karl Skretting, Jean-Marc Launay og Svend Ådne Austvoll	
Tittel på bacheloroppgaven: Depalletering av produkter ved hjelp av robot Engelsk tittel: Depalletizing products using a robot	
Studiepoeng: 20	
Emneord: Robot, OpenCv, RobotStudio	Sidetall: 44 + vedlegg/annet: 56 Stavanger 25. mai 2021

Sammendrag

Selv om Norge er Skandinavias minst robotiserte land har Norsk industri et stort potensiale til å øke robottettheten da store deler av arbeidet i bedrifter er manuelt. Norsk industri er avhengig av robotisering for å kunne opprettholde konkurranseevnen mot lavkostland. Bruken av roboter i automatisering av prosesser har de siste årene økt i Norge. Norges matindustri er blant dem som er godt i gang med bruken av roboter.

RobotNorge AS er landets ledene robotiseringsselskap og de jobber med å utvikle robotløsninger for fremtidens produksjonsbehov. En viktig kunde for RobotNorge er TINE SA. Samarbeidet mellom de to bedriftene har ført til økt produktivitet i meierier ved hjelp av robotisering.
For tiden samarbeider RobotNorge og TINE med et prosjekt kalt Skolelyst. Prosjektet har som mål å helautomatisere en pakkelinje som skal sørge for ferdigpakkede klassesett til barneskoler. Frem til nå pakkes de bestilte varene manuelt i poser. RobotNorges oppgave blir å automatisere prosessen ved å bruke ABB roboter istedenfor operatører. Systemet skal flytte produktene fra en palle til et samlebånd for å så pakke bestillingene til hver klasse i poser.

Rapporten vil ta for seg den første delen av Skolelyst prosjektet, hvor roboten skal plukke produkter fra en palle og flytte disse over til et samlebånd. Roboten vil bevege seg på en gulvbane slik at den kan nå frem til hver palle langs banen. Når roboten står foran en palle vil den først ta et bilde av hele pallen fra siden før den så roterer griperen og tar et nytt bilde sett ovenfra. For å hente ut informasjon av de to bildene gjennomføres en bildebehandling av bildene i OpenCV. Deretter vil roboten plukke et pappbrett med 12 produkter og plassere disse på et samlebånd. Når pallene begynner å tømmes vil det en gå en varsling til operatører slik at de kan erstatte nye fulle paller med de som er tomme. For å simulere prosessen ble RobotStudio og RAPID brukt. En simulering av prosessen brukes til å visualisere og teste hvordan roboten vil utføre arbeidet og eventuelt finne feil og ting som kan forbedres. Videoen kan sees ved å følge linken til videoen, link.

Konklusjonen er at selve bildebehandlingen og simuleringen av prosessen ble vellykket, men det var ingen mulighet for å samkjøre dette grunnet mangel på robot.

Forord

Forfatterne av denne rapporten er Mari Amdalsrød Hognestad og Sander André Søndeland som begge studerer Bachelor i automatisering og elektronikkdesign ved Institutt for Data- og Elektroteknikk.

Vi har i løpet av de siste fire månedene samarbeidet med RobotNorge i forbindelse med et prosjekt som de har med TINE. Prosjektets omfang er komplekst, så vår rapport omhandler kun en mindre del av prosjektet. Vår problemstilling går ut på å depalletere produkter som kommer på en pall og over til et transportbånd for videre prosessering. Rapporten gir en beskrivelse av de viktigste elementene som vi har gjennomgått, hvordan vi løste oppgaven og hva resultatet ble.

Vi ønsker spesielt å rette en takk til de som har vært sentrale i utviklingen av prosjektet.

- **Karl Skretting** for god oppfølging og teoretisk veiledning i forbindelse med prosjektet og rapporten.
- **RobotNorge AS** v/ Jean-Marc Launay og Svend Ådne Austvoll for muligheten til å jobbe med et interessant prosjekt, samt god veiledning underveis i prosjektet.

Forkortelser og begreper

Beskrivelsene er hentet fra ELE610 [21].

- Robot: En fysisk robot
- Kontroller: Dette er styreskapet som inneholder elektronikk og programvare som er koblet opp mot roboten.
- System: System består av roboten med styreskap og programvare.
- Stasjon: I RobotStudio er en stasjon en modell av den fysiske roboten. Stasjonen inkluderer omgivelsene, verktøy og arbeidsstykker knyttet til roboten.
- Virtual Controller: Dette er en virtuell versjon av kontroller som gjør at en kan simulere bevegelser og styre roboten.

- Solution: En løsning i RobotStudio består av både en stasjon, en eller flere virtuelle kontrollere og program. Dette tilsvarer en modell av systemet i RobotStudio.
- Robot-verktøy: Utstyr montert på robotens verktøyfeste som gjør roboten i stand til å utføre ønskede oppgaver.
- Smart Component: Smart Komponent går ut på at en kan lage egne verktøy og gi disse ulike funksjoner. For eksempel kan aktivere sugekopper slik at de fungere tilsvarende som de ville gjort i virkeligheten.
- ELE610: Emnekoden til *Praktisk robotteknikk* som undervises på Universitetet i Stavanger.
- DAT110: Emnekoden til *Grunnleggende programmering* som undervises på Universitetet i Stavanger.

Innhold

Kapittel 1

Introduksjon

1.1 Motivasjon

Palletering er en operasjon hvor en gjenstand plasseres på en pall i et definert mønster. Depalletering er operasjonen hvor gjenstanden fjernes i motsatt mønster [16]. Forskjellen mellom disse er at depalletering er en mer komplekst oppgave med flere utfordringer på grunn av mønsteret ikke er kjent. Objektene vil ikke alltid ligge på teoretisk plassering, men kan variere og flytte på seg.

1.2 Robotikk anvendt i industri

Deler av norsk industri velger å automatisere prosesser ved å erstatte menneskelig arbeidskraft med selvvirkende systemer. Det er et stor omfang av systemer som kan automatiseres, hvor fordelene kan være mange. [8]

I 2018 kom *International Federation og Robotics* (IFR) ut med en rapport som blant annet viser antall industriroboter, også kalt robottetthet, i verden. I følge rapporten har Norge en robottetthet på 51, som vil si at Norge har 51 industrielle roboter per 10 000 ansatt. Norge ligger derfor under gjennomsnittet som ligger på 74 roboter per 10 000 ansatte og er det minst robotiserte landet i Skandinavia [6].

Norsk industri er avhengig av robotisering for å kunne opprettholde konkurranseevnen mot lavkostland. Bruken av robotisering og automatisering har som nevnt flere fordeler [7].

Roboter kan brukes til å forenkle rutinemessige oppgaver. Ettersom roboten jobber i et jevnt tempo, vil det føre til økt effektivitet og tidbesparelser. I tillegg vil dette resultere i punktlighet fordi prosessen vil alltid utføres til samme tid. Siden robotiseringen vil ta seg av de tidkrevende og manuelle oppgavene vil dette igjen medføre at den som tok seg av oppgaven tidligere kan heller fokusere på arbeidsoppgaver som krever mer menneskelig intelligens.

Bruken av roboter vil også eliminere sjansen for menneskelige feil. Det kan selvfølgelig også oppstå feil i automatiserte prosesser men prosessen vil fortsatt bli mer nøyaktig i tillegg vil en feilmelding bli rapportert tilbake. [10] Kort oppsummert vil robotisering kutte kostnader og ressurser ved at ansatte slipper å bruke unødvendig tid på administrative oppgaver [11].

1.3 TINE SA

TINE er en norsk merkevareleverandør som daglig leverer produkter over hele Norge [13]. De har et ønske om at skolebarn skal få tilbud om god og næringsrik mat og drikke i løpet av skoledagen. Derfor har de startet en skolemelkordning, kjent som skolelyst, hvor elever kan bestille skolemelk hele året. Per dags dato, er det operatører som pakker de bestilte varene manuelt. TINE ønsker å robotisere denne prosessen slik at ordningen blir kostnadseffektiv og hurtig.

1.4 RobotNorge AS

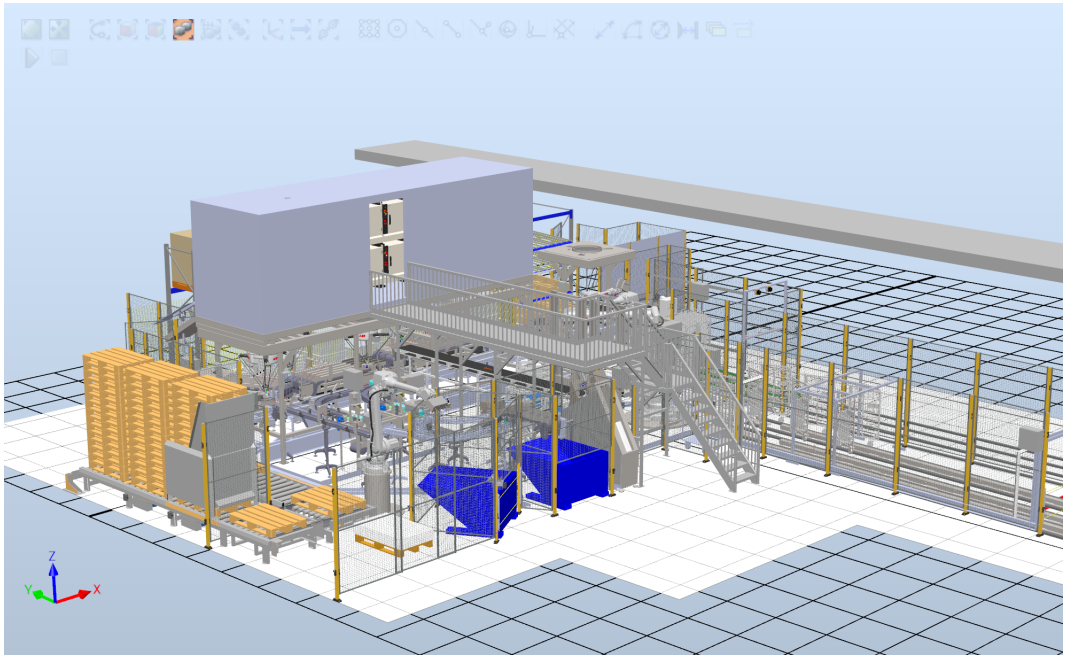
RobotNorge ble etablert i 2003 og er i dag landets ledende robotiseringsselskap. Bedriften er eneforhandler av ABB roboter i Norge og utvikler robotløsninger for fremtidens produksjonsbehov. De kombinerer ny teknologi med industrielle ABB roboter for å skape lønnsom og bærekraftig produksjonsindustri i Norge. RobotNorge eies av Trolltunga AS og har et søsterselskap, RobNor AB, i Sverige [17].

TINE har vært en viktig kunde for RobotNorge siden slutten av 90-tallet da de leverte sitt første robotiseringsposjekt på Jæren. Gjennom årene har samarbeidet mellom de to bredriftene ført til økt produktivitet i meierier ved hjelp av robotisering [9].

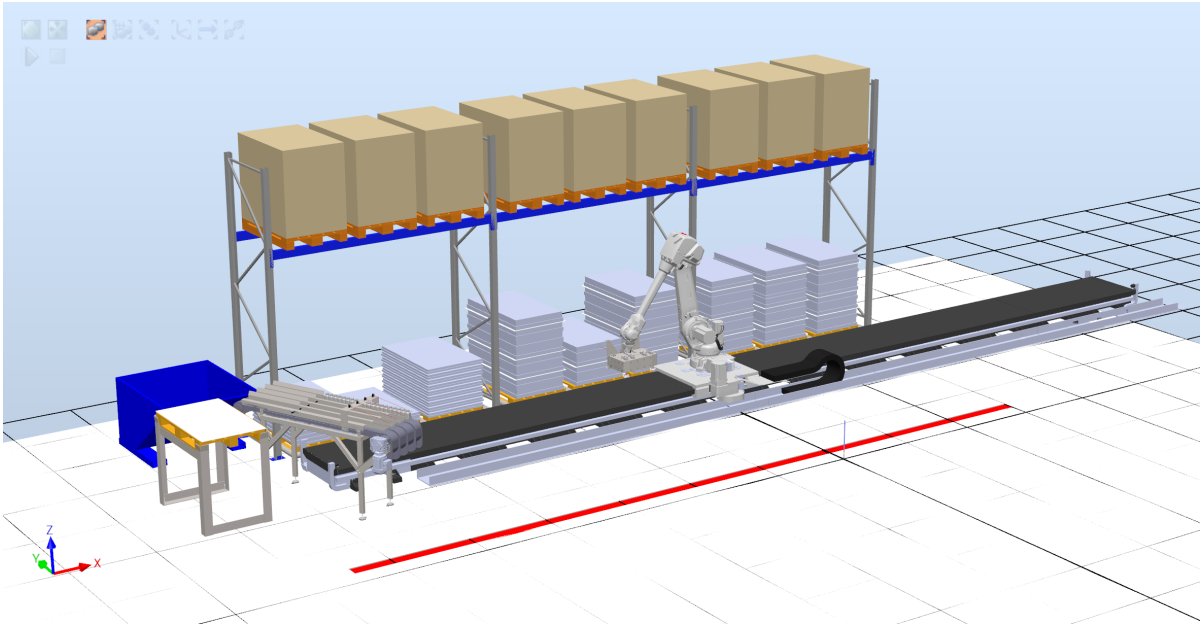
RobotNorge har tatt på seg oppgaven å utvikle et system som skal robotisere pakkeprosessen til TINE, forklart i delkapittel 1.3. Systemet skal flytte produkter fra en palle til et samlebånd for å så videre pakke bestillingene til hver klasse i poser ved hjep av ABB roboter istedenfor operatører. For en oversikt over de ulike produktene se vedlegg 6.3. Målet er at automatiseringen skal gi TINE muligheten til å effektivisere hele pakkeprosessen samtidig som den vil gi høyere fleksibilitet og bedre HMS [9].

1.5 Oppgavebeskrivelse

Dette er en bacheloroppgave med tittelen *Depalletering av produkter ved hjelp av robot*. I samarbeid med RobotNorge vil denne oppgaven ta for seg en del av Skolelyst prosjektet. Oppgaven omhandler en depalleteringsprosess, hvor en robot er brukt til å flytte produkter fra en palle og over til et samlebånd. Produktene som skal depalleteres i dette prosjektet er vist i artikkel 4529 i vedlegg 6.3. For mer informasjon om produktene se delkapittel 2.4.



Figur 1.1: Bildet viser prosjektet i sin helhet. Dette er laget av RobotNorge som et oppdrag for Tine SA.



Figur 1.2: Delen av Tine-prosjektet som representere denne oppgaven.

Figur 1.2 viser miljøet rundt roboten. Roboten vil bevege seg på en 12 meter lang gulvbane, hvor gulvbanen vil fungere som en syvende akse slik at den kan nå frem til hver palle. Pallene vil derfor bli plassert langs robotens gulvbane, hvor hver palle vil ha en fast plassering og vil kun bestå av ett produkt. Figur 1.1 viser bilde av miljøet til denne oppgaven. Her kan man se at den første plasseringen vil kun få inn paller med produkt 1, den andre plasseringen vil få inn palle med produkt 2, osv. Ergo, det er ikke nødvendig å identifisere produktet på grunn av at produktene har fast plassering. Mer

om prosessen kommer i delkapittel 3.1.

1.6 Justeringer i oppgaven

I starten av prosjektet ble det tatt valg om hvordan oppgaven, forklart i delkapittel 1.5, skulle løses. Dette delkapittelet vil ta for seg de justeringene som ble gjort.

1.6.1 Hvordan lokalisere plasseringen til produktene

Objektdeteksjon er en datasynteknikk som gjør det mulig å identifisere og lokalisere objekter i et bilde eller en video [12]. Her vil objektdeteksjonen utføres i Python ettersom det er tidligere brukt i emnene DAT110 og ELE610. Selv om programmeringsspråket allerede var bestemt var valget om hvilket programvarebibliotek som skulle benyttes ikke fastslått. Valget stod mellom OpenCV og Tensorflow. Hovedforskjellen mellom alternativene er at Tensorflow er et rammeverk for maskinlæring og OpenCV er et bibliotek for datasyn [25]. Omsider ble det avgjort at OpenCV var det programvarebiblioteket som skulle brukes til å finne plasseringen til produktene. Tid var en faktor på avgjørelsen og ettersom Tensorflow tok lengre tid å lære enn planlagt ble det til slutt valgt bort. Derimot er OpenCV brukervennlig, i tillegg til at det også ble brukt i ELE610. OpenCV er også flittig brukt når det kommer til bildefangst og bilbehandling. For mer informasjon om OpenCV se delkapittel 2.2.

1.6.2 Simulering og design av prosessen

For å kunne sette opp et visuelt bilde av selve miljøet rundt roboten ble RobotStudio tatt i bruk. RobotStudio ble valgt da dette er noe som har blitt brukt i tidligere fag ved Universitet i Stavanger, i tillegg bruker også RobotNorge RobotStudio. RobotStudio er ABB sitt program som brukes sammen med robotene deres. Fordelene med RobotStudio er at en kan lage gode simuleringer og de kan lages ved bruk av både grafisk og tekstbasert programmering. Mer informasjon om RobotStudio finnes i delkapittel 2.3.

1.6.3 Utvalg av kamera

Valg av kamera ble gjort ut fra hva som var tilgjengelig på gitt tidspunkt og basert på tidligere erfaringer. Dermed landet valget på µEye Xs. Kameraet er et enkelt og brukervennlig kamera som lett kobles opp mot datamaskinen uten problemer og det gir klare bilder ved hjelp av innebygd autofokus. Mer om selve kameraet er forklart delkapittel 4.2.2.

Et annet alternativ kunne vært å bruke et smartkamera, for eksempel fra Omron. Det er et slikt kamera RobotNorge bruker i sin løsning av problemet. En fordel med smartkamera vil være at dette har en mindre total dimensjon enn et vanlig kamera som

man trenger andre komponenter til, som for eksempel en datamaskin. En ulempe med smartkamera kan derimot være tilgjengeligheten av et begrenset utvalg i forskjellige oppløsninger. Det er altså mye større utvalg av vanlige kamera kontra smartkamera. Datakraften er også begrenset sammenlignet med PC-systemer på grunn av et kompakt design. [24]

1.6.4 Hvordan finne avstanden fra robotverktøy til produkt

For å finne avstanden til brettene kan en enten bruke informasjon ut fra bilde og ved hjelp av et program, i dette tilfellet Python eller avstandssensor. En kan lage et program som forteller avstanden til produktene ut fra arealet på sirklene til korkene. Eventuelt kan det brukes en avstandssenor for å finne måle avstanden til produktene. For å få så nøyaktige målinger som mulig falt valget på avstandssensor.

Ved å bruke kameraet til å finne avstanden til produktene kan dette påvirkes av lysforholdene. Dette er noe som ikke har innvirkning på avstandssensoren og derfor gjør det avstandssensoren til det tryggeste valget.

1.7 Oversikt over struktur og innhold i rapporten

Oppgaven starter med informasjon om hvilke verktøy som er brukt for å realisere prosjektet. Dette består av hvilket kamera som ble brukt, program til bildefangst, RobotStudio og de forskjellige produktene. Deretter kommer en beskrivelse av vår løsning av oppgaven. Det kommer beskrivelse av deteksjon av pappbrett og lokalisering av korker og hvordan dette er bygd opp. Første tema innen RobotStudio er smart komponent. Så kommer en beskrivelse av kommunikasjon mellom Python og RobotStudio, deretter følger en beskrivelse av simuleringen i RobotStudio. Til slutt kommer oppsummeringen som inneholder tester og en diskusjonsdel.

Kapittel 2

Bakgrunn

Dette kapittelet vil ta for seg de ulike verktøyene som benyttes i forbindelse med oppgaven. Det vil kun bli en kort introduksjon, nok til å forstå teorien bak løsningen.

2.1 Kamera µEye

Valget av kamera ble tidlig bestemt til å være µEye som er tilgjengelig på universitet i Stavanger. Dette er det samme kameraet som blant annet brukes i ELE 610. Kameraet som ble brukt i oppgaven er uEye XS fra IDS. Dette er et lite kamera med blant annet autofokus som en av sine funksjoner. Dette kameraet er laget for å være et forbruker kamera som kan brukes i industrielle settinger. Oppkoblingen til kameraet er gjennom USB 2.0 interface og en Mini B USB 2.0 kontakt. Pikselstørrelsen er 1.4 µm og 5.04 megapiksler. Dette gjør at kameraet leverer god bildekvalitet og nøyaktige fargegjengivelse selv i de tøffeste lysforholdene.

uEye XS er et av de minste kameraene en finner i markedet med sin vekt på kun 12 gram og kompakte dimensjoner på 26.5 x 23 x 21.5 millimeter.

2.2 OpenCV

OpenCV er et åpent kildekode bibliotek for datasyn og maskinlæring. At openCV er en åpen kildekode betyr at det er et gratis rammeverk som alle kan laste ned og bruke. OpenCV utvikles videre ved at brukere fra hele verden sender inn tilbakemeldinger slik at det kan forbedres og lages biblioteker som en kan bruke. Det finnes mer enn 47 tusen brukere og antallet nedlastninger har passert 18 millioner. [2]

OpenCV et et bibliotek designet for å løse problemer med datasyn. Datasyn danner grunnlaget for en automatisk forståelse av digitale bilder og brukes til å konstruere systemer for maskinsyn [23]. Maskinsyn er en teknologi som gjør at en maskin kan tolke og forstå verden rundt seg ved hjelp av bildebehandling [15]. Enkelt forklart er

maskinsyn bruken av datasyn i industrielle miljøer. Uten datasyn vil ikke maskinsyn fungere ettersom det er hjernen bak behandlingen av informasjonen. [3]

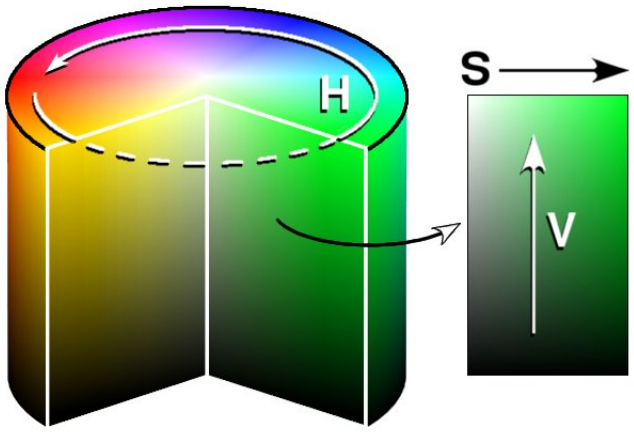
OpenCV blir brukt av store bedrifter som Google, Yahoo, Microsoft og IBM. [2]

2.2.1 Fargefiltrering - HSV

Fargefiltrering, også kjent som fargesegmentering, er ofte brukt i OpenCV for å identifisere spesifikke objekter. Det mest brukte fargerommet er RGB-fargerom, hvor primærfargene, rødt, grønt og blått legger seg sammen for å gi farge på bildet. Dersom man ønsker å identifisere en region med en bestemt farge, er det mer gunstig å bruke HSV-fargerom. Det er fordi fargene i HSV-fargerommet kan skilles lettere [20].

Figur 2.1 viser HSV fargerom som syilder. HSV står for Hue-Saturation-Value. Hue spesifiserer den dominante fargen som oppfattes av en observatør. Den er angitt som grader rundt en sirkel fra 0-360 hvor rød er 0°, grønn er 120°og blå er 240°[14].

Saturation (metningen) definerer intensiteten til fargen. Metningen måles i prosent og den kontrollerer mengden farge som brukes. Figur 2.1 viser at dersom S beveger seg mot venstre, som vil si 0% metning, vil fargen gå mot en gråtone. Dersom den beveger seg mot 100% metning vil fargen bli mye mer tydelig og renere. Valøren styrer fargens lysstyrke. Denne måles også i prosent hvor 0% lysstyrke gjør at fargen blir helt svart og 100% lysstyrke vil resultere i at fargen ikke vil innerholde noe svart [4].



Figur 2.1: HSV fargerom som sylinder [5].

Det eneste som trengs for å fargesegmentere i OpenCV er terskelverdiene om det nedre og øvre grenseområdet for fargen i HSV-fargerommet.

2.3 Verktøyet RobotStudio

RobotStudio er utviklet av ABB og fungerer som et offline simulering- og programmeringsverktøy for roboter. Verktøyet gjør det blant annet mulig å visualisere, programmere og teste simulering av robotceller- og systemer. I dette prosjektet vil derfor RobotStudio bli brukt til å designe miljøet for løsningen til oppgaven og deretter kunne teste oppsettet ved hjelp av simulering. [17]

2.3.1 Tekstbasert programmering

Det er mulig å programmere roboten ved å bruke RAPID. Det er ABBs høynivå programmeringsspråk for industriroboter. Se vedlegg 6.7 for oppsett av en Rapid kode. Etter koden er ferdig skrevet, kan programmet kjøres i en simulering og enkelt observere hvordan roboten beveger seg. Dette kan også overføres til den fysiske roboten slik at roboten beveger seg likt som simuleringen. [1]

En løsning for kommunikasjon mellom den fysiske roboten og Python på PC er laget av Markus Iversflaten og Ole Christian Handegård i 2020 og brukt i oppgave RS-5 i ELE610. Mangelen på en fysisk robot med syvende akse gjør at koblingen mellom RAPID og Python ikke kunne gjennomføres grunnet at det ikke var mulig under simulering. Se delkapittel 3.3.4 for mer nøyaktig beskrivelse av hvordan kommunikasjonen mellom Python og den fysiske roboten kan løses. [22]

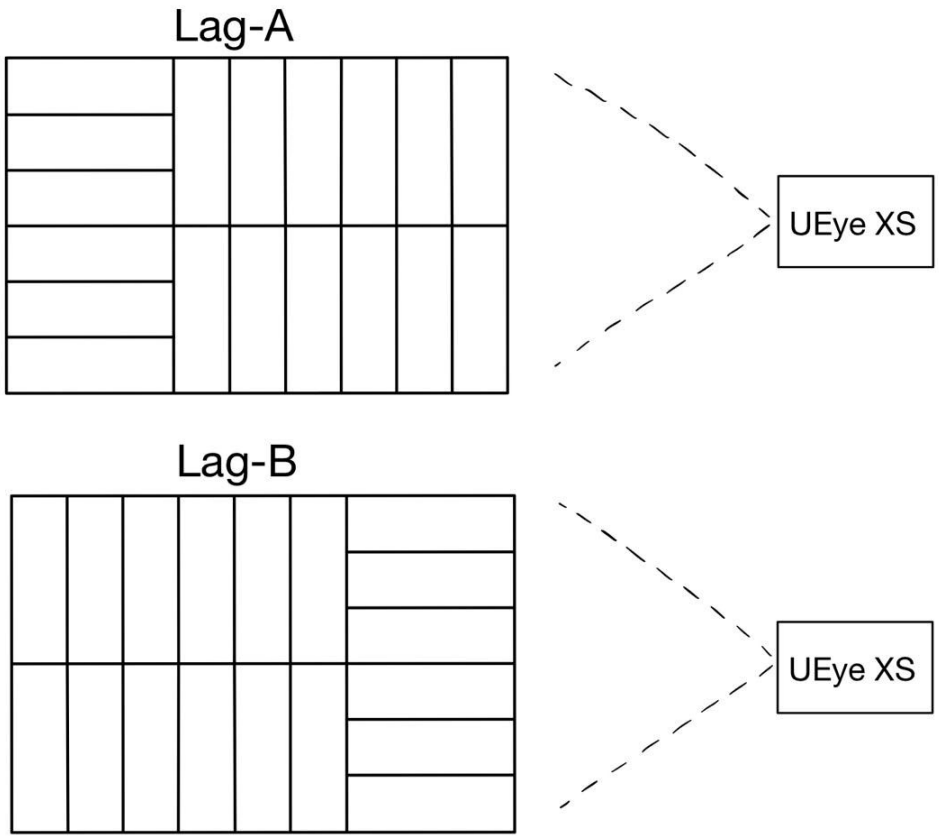
2.4 Produkter

I denne oppgaven brukes 7 ulike produkter, men siden de har samme form og er stablet likt på pallen vil det bare være nødvendig å lage løsning for et produkt. Tilsvarende løsning vil være for produktet vist i artikkel 4529 i vedlegg 6.3.

En palle består av to ulike lag som stables annen hver. Se figur 2.2 for hvordan en palle med produkter vil se ut. Hvert lag vil innerholde 216 produkter som er plassert i 18 pappbrett. Forskjellen på de to lagene er plasseringen av pappbrettene, se figur 2.3. Lag-A og Lag-B vil altså stables vekselvis. Mellom hvert lag vil det ligge en papplate som også må fjernes. Denne vil fjernes etter de 18 pappbrettene er fjernet.

Figur 2.2: Eksempel på hvordan en palle med melk vil se ut.
Foto: Jean-Marc Launay/RobotNorge

Hver palle vil bestå av en type produkt og ha en fast plassering langs robotens gulvbane. Av den grunn er der derfor ikke nødvendig å identifisere hvert produkt men heller detektere posisjonen til pappbrettet som produktene ligger i.



Figur 2.3: Bilde av Lag-A og Lag-B. Hver rute visualiserer et pappbrett.

Hvis det øverste brettet til venstre ligger på kortsiden viser figur 2.3 at det er Lag-B og dersom pappbrettet ligger på langsiden er det Lag-A. Stablingen vil derfor ha et fast mønster, enten er topplaget Lag-A og laget under er Lag-B, eller omvendt. Som følge av det trenger kun det første brettet å detekteres.

Kapittel 3

Robotløsning med maskinsyn

Dette kapittelet vil ta for seg løsningen som er foreslått og hvordan den virker. Det vil komme en redegjørelse for hvordan brettene detekteres og hvordan roboten vil motta informasjonen.

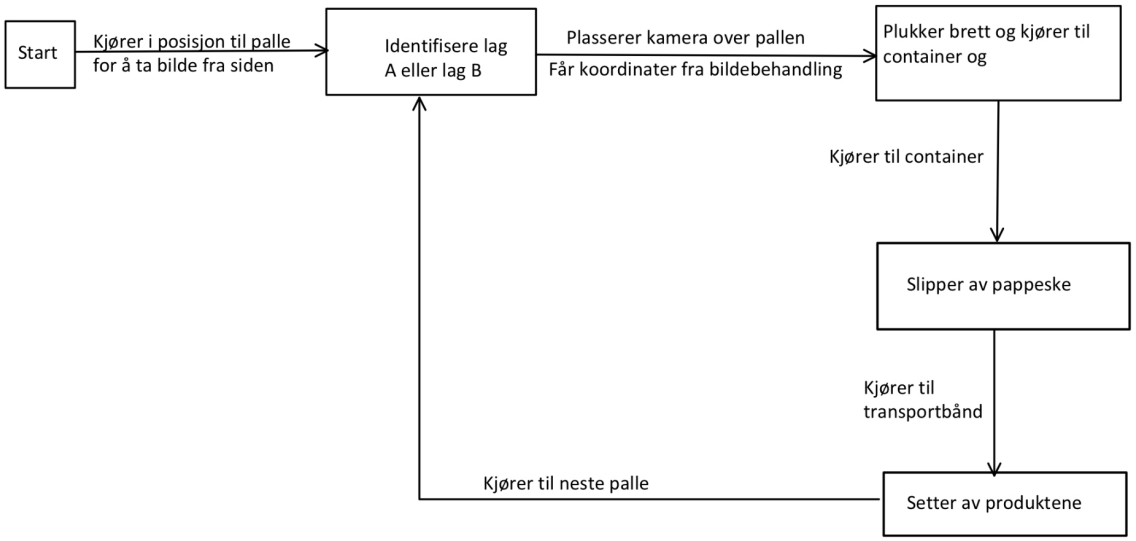
Tilgangen til paller og pappbrett med produkter var begrenset hele perioden. Derfor ble det kun brukt to pappbrett i denne løsningen. Pappbrettene ble plassert slik som de ville ligget på pallene. Dette var for å komme så nær realistisk plassering som mulig. Figur 3.2 og figur 3.1 viser de to ulike måtene brettene kan ligge på. Det er disse bildene som vil bli brukt til løsningen av denne oppgaven.

Figur 3.1: Hypotetisk plassering av brett på en palle ved Lag-A.

Figur 3.2: Hypotetisk plassering av brett på en palle ved Lag-B.

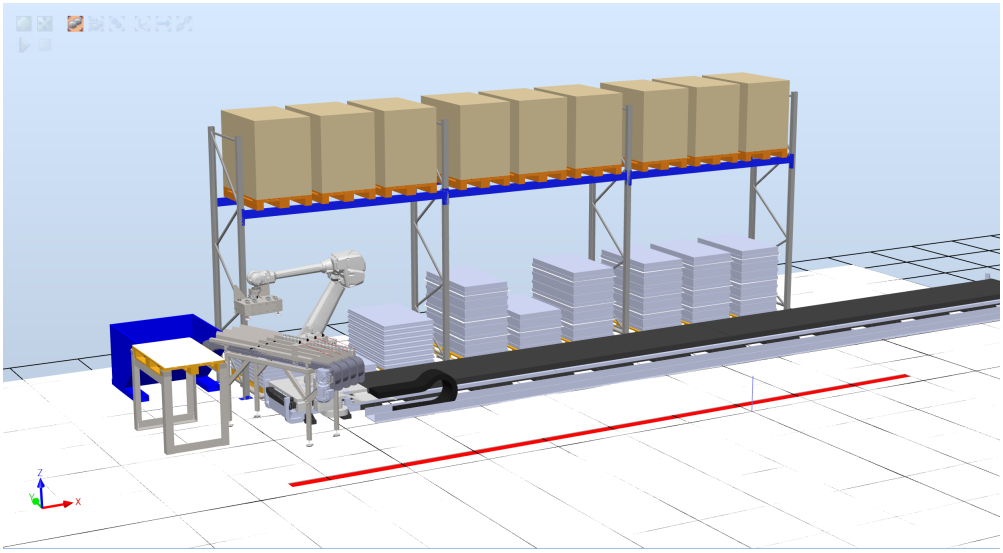
3.1 Om prosessen

En forklaring på hvordan roboten vil kjøre er vist i figur 3.3.



Figur 3.3: Tilstandsdiagram som viser mønsteret til roboten i prosessen.

Figur 3.4 viser posisjonen til roboten ved start. Denne posisjonen er en oversiktsposisjon og kalles for posisjon 1. Når prosessen starter vil roboten kjøre fra posisjon 1 til første og nærmeste palle. Når roboten står foran pallen vil den ta et bilde fra siden. Etter bildet er behandlet vil roboten rotere griperen slik at den står over pallen. Her vil den ta ett nytt bilde sett ovenfra.



Figur 3.4: Robot i posisjon 1.

Analyseringen fra begge bildene vil indikere posisjonen til det første pappbrettet som skal plukkes av roboten. Etter roboten har hentet det første pappbrettet vil den kjøre tilbake til posisjon 1. Når roboten er tilbake til posisjon 1 vil den først fjerne pappbrettet rundt produktene ved å legge det i den blå konteineren. Se figur 3.4. Nå står roboten igjen med 12 produkter som er festet til hver sin sugekopp. Disse 12 produktene vil til slutt legges på et transportbånd til videre prosessering. Se delkapittel 3.3.2 for mer informasjon om hvordan griperen fungerer.

Deretter vil roboten kjøre til neste palle og gjør samme prosedyre. Etter roboten har tatt ett pappbrett fra hver palle, vil den gå tilbake til den første pallen for å ta neste pappbrett. Etter alle pappbrettene i et lag på pallen er flyttet vil roboten fjerne papplaget som ligger mellom hvert lag. Papplaget fjernes ved at roboten fester sugekoppene til papplaget. Deretter legges det i den samme konteineren som pappbrettene blir lagt. Prosessen fortsetter til pallene begynner å tømmes. Da vil det gå en varsling til operatører slik at de kan erstatte nye fulle paller med de som er tomme.

3.2 Bildebehandling i OpenCV

Prosessen for bildebehandling er delt i to hoveddeler - deteksjon av pappbrett og deteksjon av korker. Først vil roboten ta et bilde fra siden av hele pallen med produkter. Bildet som returneres vil enten bli som figur 3.2 eller som figur 3.1. Hensikten her blir derfor å finne ut hvordan pallen er stablet, altså om det øverste laget er Lag-A eller Lag-B. Dersom bildet blir som figur 3.1 vil det øverste laget være Lag-A og hvis bildet blir likt som figur 3.2 er det øverste laget Lag-B.

Samtidig som det første bildet analyseres vil roboten gå videre til neste del av prosessen og roterer griperen slik at den er klar til å ta et bilde ovenfra. Innen den tid vil det første bildet være ferdig behandlet og som følge av det vet nå roboten hvilket lag som

ligger på toppen. Deretter vil roboten ta bilde sett ovenfra for å få en oversikt over alle korkene til produktene i det første laget. Figur 3.5 viser et bilde av korkene dersom det øverste laget er Lag-B og figur 3.6 viser et bilde av korkene dersom det øverste laget er Lag-A.

Figur 3.5: Hypotetisk plassering av korker ved Lag-B.

Figur 3.6: Hypotetisk plassering av korker ved Lag-A.

Opprinnelig ville det totalt vært 216 korker i bildet men ettersom tilgangen til pappbrett var begrenset vises det kun 24 korker i figur 3.5 og figur 3.6. Oppgaven vil fortsatt være den samme, som er å finne de 12 korkene som tilhører det første brettet som skal plukkes. Etter korkene er detektert vil posisjonen til hvert senter av korkenes kontur sendes til roboten.

3.2.1 Deteksjon av pappbrett

I delkapittel 2.4 forklares det hvordan pallene er stablet. Det er altså to lag, Lag-A og Lag-B, som stables annen hver. Oppgaven blir da å finne hvilket lag som ligger på

toppen. Etter dette er avdekket vil resten av stablingen på pallen være kjent. Det er derfor ikke nødvendig å detektere de resterende pappbrettene på pallen, men det kan være nyttig i tilfelle det er feil eller mangler på pallen. Feks. et brett som mangler eller som ikke ligger på teoretisk plassering.

For å finne det øverste laget på pallen må roboten ta et bilde av pallen fra siden for å så se hvilken retning pappbrettet øverst til venstre har. Her finnes det kun to utfall, enten ligger brettet på langsiden eller kortsiden. Figur 2.3 viser at dersom brettet ligger på langsiden er laget Lag-A og hvis det ligger på kortsiden er laget Lag-B.

For å kunne detektere pappbrettene ble det først utført HSV fargefiltrering. Filtreringen vil kunne eliminere vekk støy på bildet som ikke har samme fargeintensitet som pappbrettene. Mer om HSV fargefiltrering forklares i delkapittel 2.2.1.

Måten filtreringen utføres på er ved å sette inn verdier for H, S og V som tilsvarer samme fargeintensitet som fargen på pappbrettet. For å finne de rette HSV verdiene ble det laget en trackbar, se figur 3.7. En trackbar er et GUI element som lar brukeren velge en spesifikk verdi innenfor et verdiområde ved å skyve glidebryteren lineært. [19] Når glidebryteren skyves vil det bli satt en ny verdi og bildet vil endre seg parallelt med den gitte verdien. Derfor gjør bruken av GUI det enklere å finne riktige grenser fordi det er mulig å sammenligne med bildet samtidig.

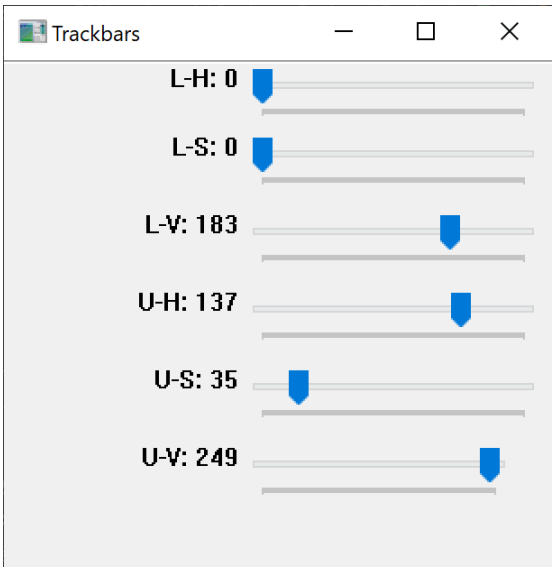
Listing 3.1 viser koden som ble brukt til å lage trackbaren. Her er verdiområdet til H [0,180] og verdiområdet til S og V er [0,255].

```
cv2.namedWindow("Trackbars")
cv2.createTrackbar("L-H", "Trackbars", 0, 180, nothing)
cv2.createTrackbar("L-S", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("L-V", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("U-H", "Trackbars", 180, 180, nothing)
cv2.createTrackbar("U-S", "Trackbars", 255, 255, nothing)
cv2.createTrackbar("U-V", "Trackbars", 255, 255, nothing)

while True:
    hsv_frame = cv2.cvtColor(resized, cv2.COLOR_BGR2HSV)

    l_h = cv2.getTrackbarPos("L-H", "Trackbars")
    l_s = cv2.getTrackbarPos("L-S", "Trackbars")
    l_v = cv2.getTrackbarPos("L-V", "Trackbars")
    u_h = cv2.getTrackbarPos("U-H", "Trackbars")
    u_s = cv2.getTrackbarPos("U-S", "Trackbars")
    u_v = cv2.getTrackbarPos("U-V", "Trackbars")
```

Listing 3.1: Kode for å lage en trackbar.



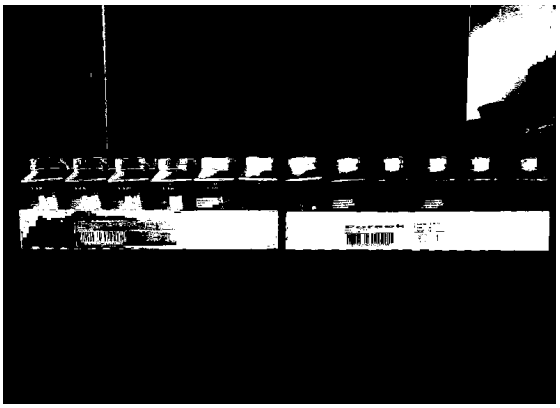
Figur 3.7: Resultat fra trackbar kode med det øvre og nedre grenseområdet for HSV-fargerommet til pappbrettene.

Verdiene som er satt i figur 3.7 er de verdiene som ble brukt i koden for filtrering, se listing 3.2. Koden for filtrering ble utført på både figur 3.2 og figur 3.1. Resultatet av filtreringen på begge bilder vises i figur 3.8. Her er det også lagt inn et testbilde hvor pappbrettene ikke ligger på teoretisk plassering men at det ene brettet er plassert som Lag-A og det andre som Lag-B. Hensikten med testbildet er å kvalitetsjekke resultatet av filtreringen.

```
hsv_frame = cv2.cvtColor(resized, cv2.COLOR_BGR2HSV)
low = np.array([0,0,183])
high = np.array([137,35,249])
mask = cv2.inRange(hsv_frame, low, high)
```

Listing 3.2: Kode hvor verdiene for filtreringen er lagt inn.

(a) Lag-A uten filtrering

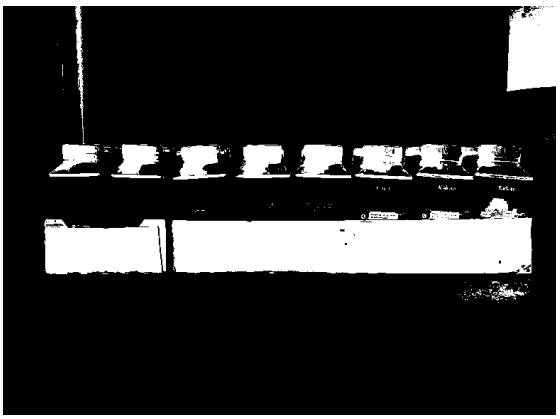


(b) Lag-A med filtrering

(c) Lag-B uten filtrering

(d) Lag-B med filtrering

(e) Testbilde uten filtrering



(f) Testbilde med filtrering

Figur 3.8: Resultat av filtrering.

Etter bildet er ferdig filtrert kan pappbrettene markeres ved å bruke funksjonen *cv2.findContours*. Funksjonen vil detektere en kontur langs grensen til et bilde som har samme intensitet og siden bildet allerede er filtrert vil den enkelt kunne markere pappbrettene.

Problemet er at selv om bildet er filtrert vil det fortsatt kunne oppstå støy, se figur 3.9. Her er det ikke bare pappbrettene som har samme intensitet, men støy er også blitt tatt med. For å fjerne støyet må det legges inn arealgrenser på konturen. Brettenes areal er

vesentlig større en enn støyet rundt, derfor legges det inn en nedre grense på konturens areal slik at all støy under grensen ikke blir detektert.

(a) Lag-A (b) Lag-B

Figur 3.9: Resultat uten grenseverdier.

Listing 3.3 viser koden som ble brukt til å finne konturen. Her er arealgrensen er satt til 10000 i linje 9. I samme kode brukes funksjonen *cv2.boundingRect* til å markere områder med en kontur ved å tegne et rektangel rundt. Funksjonen returnerer fire verdier, x,y,w og h. Hvor (x,y) er koordinatene øverst til venstre for rektangelet og (w,h) er bredden og høyden. For å være sikker på at all støy i bildet er utelukket ble det også satt øvre og nedre grenser på disse fire verdiene (x,y,w,h).

Linje 6 og 7 i listing 3.3 viser hvor grenseverdiene til høyden og bredden ble lagt inn. Gensene ble funnet ved å printe ut størrelsen på bredden og høyden til alle konturene i bildet, inkludert støy. Som følge av dette får man en oversikt over alle størrelsene i bildet og det er enklere å resonnere seg frem til de rette grenseverdiene.

```
1  contours, hierarchy = cv2.findContours(mask, cv2.RETR_TREE,
2                                         cv2.CHAIN_APPROX_SIMPLE)
3
4  for cnt in contours:
5      x, y, w, h = cv2.boundingRect(cnt)
6      if w > 50 and h > 30:
7          if w < 800 and h < 800:
8              size = cv2.contourArea(cnt)
9              if size > 10000:
10                 rek = cv2.rectangle(resized,(x,y),(x+w,y+h),(0,215,255)
       ,2)
```

Listing 3.3: Kode for å tegne kontur rundt pappbrettene med grenseverdier.

Resultatet fra listing 3.3 vises i figur 3.10. Her er begge pappbrettene i hvert lag detektert og alt støy rundt er neglisjert.

(a) Lag-A (b) Lag-B

Figur 3.10: Resultat med grenseverdier.

Neste oppgave er å lage kontur rundt det første pappbrettet til venstre og fjerne konturen til den andre. Ettersom roboten vil ta bilde fra samme plass er det mulig å sette inn begrensninger ved å legge inn bestemte koordinater. Ved å ta bilde fra denne posisjonen vil det første hjørnet, sett fra venstre, treffe omtrentlig på x aksen rundt x = 85. Dersom dette settes inn i koden vil alt utenfor dette området neglisjeres. Begrensningen på x aksen ble lagt inn i første linje i listing 3.4. Her er ikke x lik 85 men den er satt til mindre enn 111. Dette er for å lage en sikkerhetsmargin i tilfelle pallens plassering skulle kommet utenfor det området det var ment å bli å bli plassert.

Det er det ikke nødvendig å legge inn begrensinger på y aksen ettersom det ikke ligger et lag under, men dersom dette var tilfellet legges dette inn sammen med grensen til x aksen. Etter begrensingene er lagt inn blir resultatet likt som i figur 3.11.

(a) Lag-A (b) Lag-B

Figur 3.11: Kontur av et brett.

Etter det første brettet er lokalisert er det neste som gjenstår å finne ut om det første laget er Lag-A eller Lag-B. Derfor sjekkes arealet på de to brettene. Naturligvis vil brettet som ligger på langsiden ha et større areal enn brettet som ligger på kortsiden. Ved å printe ut arealet på de to brettene ser man at det minste arealet ligger på rundt 10000 mens det største arealet ligger på over 13000. Av den grunn ble det lagt inn en arealbegrensning på 12000 i koden, se listing 3.4. Koden viser at dersom arealet er større enn 12000 vil brettet ligge på langsiden og dermed returnere Lag-A. Hvis ikke vil Lag-B returneres.

```
if x < 111:
    if size > 12000:
        rek = cv2.rectangle(resized,(x,y),(x+w,y+h),(0,215,255),2)
        print('Lag-A')
        Lag = 1
    else:
        rek = cv2.rectangle(resized,(x,y),(x+w,y+h),(0,215,255),2)
        print('Lag-B')
        Lag = 0
```

Listing 3.4: Kode for å finne ut om laget er Lag-A eller Lag-B.

3.2.2 Deteksjon av korker

Når mønsteret på pallen er definert vil roboten ta et bilde ovenfra. Bildet vil da vise korkene til produktene, se figur 3.5 og figur 3.6. Neste del av oppgaven blir da å spesifisere hvilke 12 produkter som skal plukkes opp først. Siden roboten allerede har funnet posisjonen til pappbrettet som ligger øverst til venstre, vil det da være naturlig å velge de 12 produktene som tilhører det brettet.

Løsningen er derfor avhengig om det øverste laget er Lag-A eller Lag-B. Av den grunn ble det lagt inn i koden i listing 3.4 at dersom laget er Lag-A settes Lag=1 og dersom laget er Lag-B settes Lag=0 for å skille de to utfallene.

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray_blurred = cv2.blur(gray, (3, 3))

circles = cv2.HoughCircles(img = gray_blurred, method = cv2.
    HOUGH_GRADIENT, dp = 1, minDist = 20, param1 = 50, param2 = 30,
    minRadius = 23, maxRadius = 25)
```

Listing 3.5: Kode for deteksjon av sirkler.

Korkene er formet som sirkler og for å kunne detektere sirklene må funksjonen *cv2.HoughCircles* benyttes. Se listing 3.5. Funksjonen tar inn flere parametere. Under kommer en kort forklaring på hver parameter som er hentet fra [18].

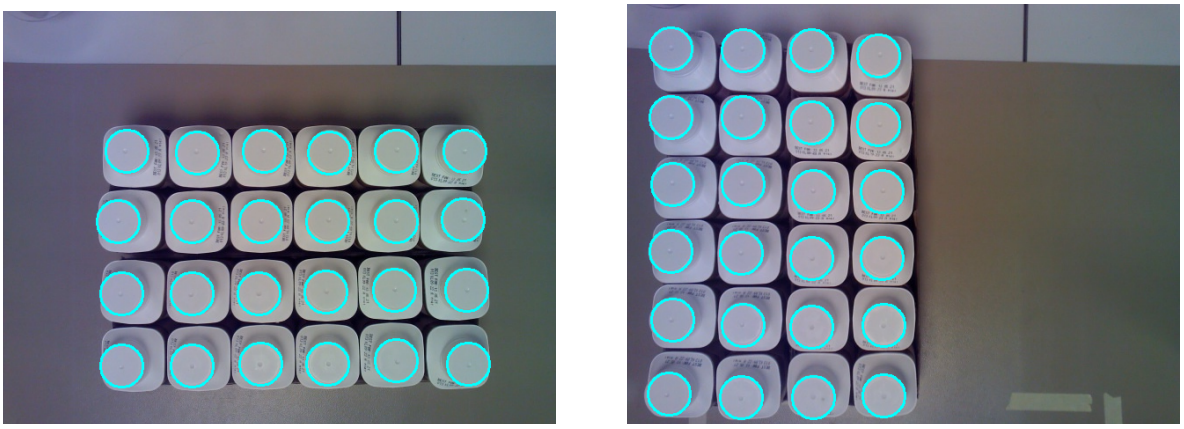
- **img** - tar inn et 1-channel bilde. Ettersom det originale bildet er et fargebilde, som vil si 3-channel, må det omformes til gråskala. I listing 3.5 brukes funksjonen cv2.COLOR_BGR2GRAY til omformingen.

- **method** - definerer metoden for å oppdage sirkler i bildet. Her er cv2.HOUGH_GRADIENT foreløpig den eneste implementerte metoden.
- **dp** - forteller om oppløsning av akkumulatorarray. Det vil si at dersom denne settes lav vil den detektere nesten perfekte sirkler. Kontra om den settes for høyt vil den ta imot støy og detektere sirkler hvor det ikke eksisterer sirkler.
- **minDist** - er minste avstanden mellom sentrum til hver detekterte sirkel.
- **param1** - gradientverdi brukt til å håndtere kantdeteksjon.
- **param2** - akkumulatorterskelverdi for metoden cv2.HOUGH_GRADIENT. Hvis denne settes for lav kan flere falske sirkler detekteres.
- **minRadius** - minimum størrelse på radiusen gitt i piksler.
- **maxRadius** - maksimal størrelse på radiusen gitt i piksler.

Før sirklene kan detekteres må bildet filteres. I listing 3.5 blir bildet først gjort om til gråskala og deretter glattes det ut. Etter dette er gjort må konturene lages slik at de vises på bildet, se listing 3.6. Koden starter med å forsikre seg om at det er noen sirkler som er detektert. Deretter lager den en løkke over (x,y) koordinatene og radiusen til sirklene hvor den til slutt tegner sirkelen inn i bildet.

```
1 if circles is not None:
2     circles = np.round(circles[0, :]).astype("int")
3     for (x, y, r) in circles:
4             a = cv2.circle(output, (x, y), r, (0, 255, 255), 2)
```

Listing 3.6: Kode for å tegne kontur rundt korkene.



(a) Lag-A (b) Lag-B

Figur 3.12: Kontur rundt alle korker.

Som resultat fra listing 3.6 vil alle korkene på bildet få en kontur rundt seg. Se figur 3.12. Neste oppgave blir å spesifisere hvilke 12 sirkler som skal detekteres og dermed plukkes opp av roboten. Dette løses ved å sette begrensninger for x og y koordinater i bildet. Her vil koordinatene bestemmes ut fra resultatet fra delkapittel 3.2.1, altså om laget er Lag-A eller Lag-B.

Dersom resultatet fra delkapittel 3.2.1 viser at det første laget er Lag-B, vil bildet roboten tar ovenfra bli som figur 3.12b. De første produktene som roboten da skal plukke opp er de som ligger innenfor rektangelet markert i rødt i figur 3.13. For å gjøre dette må det lages begrensninger på x og y aksen i bildet. Se listing 3.7 for hvordan det ble gjort.

Figur 3.13: Det markerte området er det pappbrettet som skal plukkes først i Lag-B.

```
        for (x, y, r) in circles:
            if min_x < x < max_x and min_y < y < max_y:
                b = cv2.circle(output, (x, y), r, (255, 255, 4), 2)
```

Listing 3.7: Kode for begrensning av x og y aksen i bildet.

Deretter ble det testet med ulike grenseverdier for å finne de rette verdiene. Testene viste at verdiene i tabell 3.1 var de verdiene som detekterte de ønskede korkene. Figur 3.14 viser resultatet etter verdiene ble lagt inn.

max_x	150
min_x	0
max_y	450
min_y	50

Tabell 3.1: Grenseverdier for første lag i Lag-B.

Figur 3.14: Første pappbrett som skal plukkes i Lag-B.

Nå som de rette sirklene er detektert må koordinatene til sirklene sendes til roboten. Dette ble gjort ved å finne sentrum til sirklene og printe ut disse, se listing 3.8.

Figur 3.15a viser koordinatene til de detekterte sirklene i figur 3.14 samt radiusen. Disse koordinatene vil bli sendt til roboten slik at den vet hvor den skal plukke produktene.

```
        for i in circles:
            if min_y < i[1] < max_y and min_x < i[0] < max_x:
                b = cv2.circle(output,(i[0],i[1]),2,(0,0,255),3)
                print(i)
```

Listing 3.8: Kode for å finne sentrum til sirkler.

```
[ 54 360  24]
[134 366  24]
[ 50 444  24]
[132 448  24]
[52 52 24]
[ 54 206  24]
[ 52 282  24]
[132 284  24]
[134 206  24]
[134  54  24]
[ 52 130  24]
[134 130  24]
```

(a) Første pappbrett

```
[294 366  24]
[294 446  24]
[212 368  24]
[212 290  24]
[214 448  24]
[288 138  24]
[212 136  24]
[214 214  24]
[292 290  24]
[288  60  24]
[212  54  24]
[292 214  24]
```

(b) Andre pappbrett

Figur 3.15: Koordinatene til pappbrett i Lag-B.

Etter det første pappbrettet med produkter er detektert og plukket opp av roboten, skal neste pappbrett detekteres. Siden pappbrettene vil bli plassert på pallen med et fast mønster, kan neste pappbrett detekteres ved å endre på grenseverdiene til x og y. Det neste pappbrettet vil altså ligge ved siden av det første pappbrettet i figur 3.14.

De samme testene som ble gjort for å finne verdiene i tabell 3.1 ble også gjort for å finne grenseverdiene til det neste pappbrettet. Disse er vist i tabell 3.2.

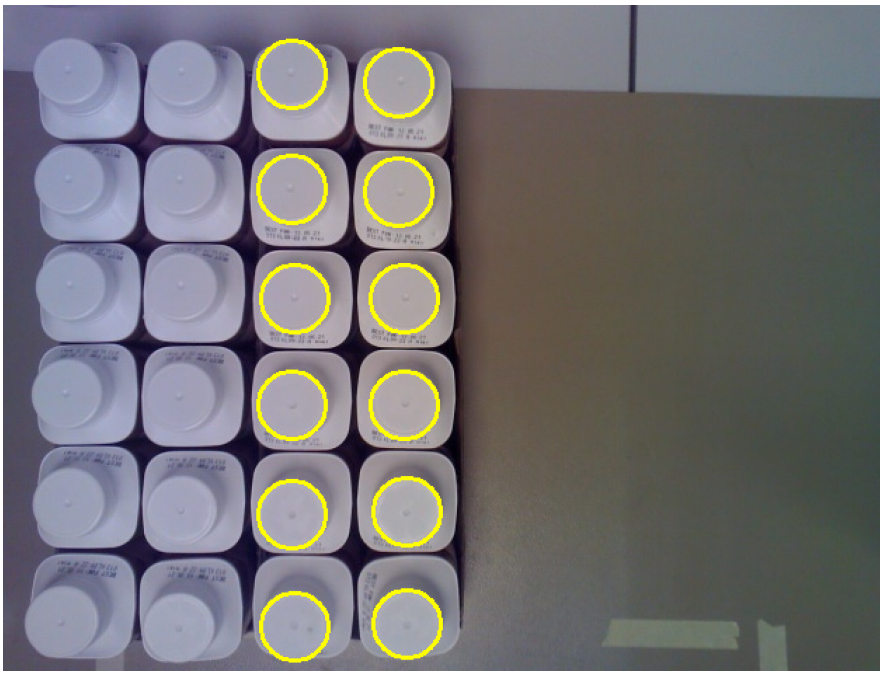
max_x	300
min_x	150
max_y	450
min_y	50

Tabell 3.2: Grenseverdier for neste brett i Lag-B.

Istedenfor å legge inn verdiene fra tabell 3.2 inn i koden, brukes heller differansen mellom tabell 3.2 og tabell 3.1. På denne måten vil roboten kunne plukke neste brett uten å måtte gi verdier. Det er fordi avstanden mellom produktene vil omtrentlig være den samme, så hvis det hadde vært et tredje pappbrett ville også den blitt detektert i neste runde. Koden for deteksjon av neste pappbrett er vist i listing 3.9. Resultatet fra listing 3.9 vises i figur 3.16 og koordinatene som sendes tilbake til roboten er vist i figur 3.15b.

```
for (x, y, r) in circles:
    if (min_x + 150) < x < (max_x + 150) and min_y < y < max_y:
        b = cv2.circle(output, (x, y), r, (255, 255, 4), 2)
for i in circles:
    if min_y < i[1] < max_y and (min_x + 150) < i[0] < (max_x + 150):
        b = cv2.circle(output,(i[0],i[1]),2,(0,0,255),3)
        print(i)
```

Listing 3.9: Kode for deteksjon av neste pappbrett i Lag-B.



Figur 3.16: Neste pappbrett som skal plukkes i Lag-B.

Frem til nå er det vist hvordan fremgangsmåten for deteksjon av et pappbrett med produkter i Lag-B fungerer. Dersom pappbrettet er i Lag-A er det samme kode og fremgangsmåte som for Lag-B. Eneste som må endres på er grenseverdiene til x og y aksen. Grenseverdiene til første pappbrett i Lag-A står i tabell 3.3. Tabell 3.4 viser grenseverdiene til neste pappbrett.

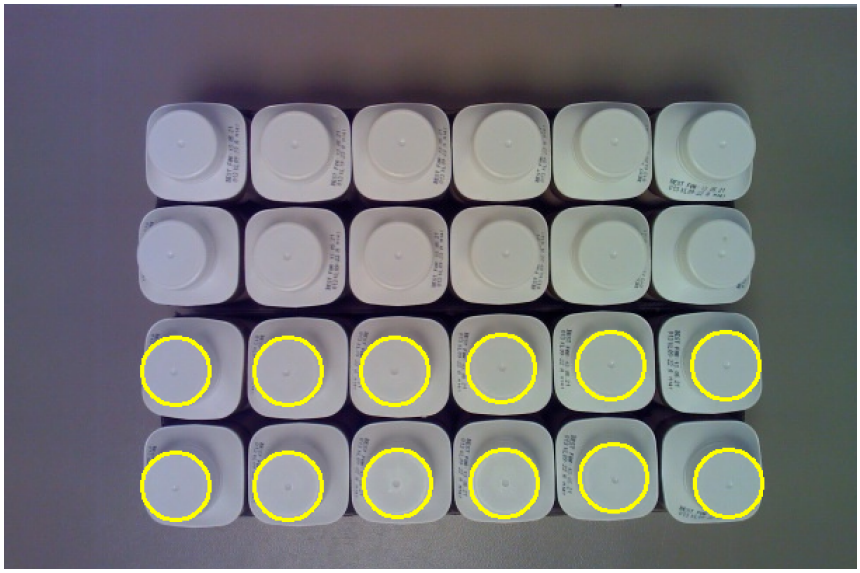
For å detektere de første 12 korkene i Lag-A, se figur 3.17, brukes koden i listing 3.7 sammen med grenseverdiene i tabell 3.3. Deretter brukes koden i listing 3.8 for å printe ut sentrum til de detekterte korkene. Figur 3.19a viser koordinatene til de detekterte korkene i figur 3.17.

max_x	550
min_x	100
max_y	450
min_y	300

Tabell 3.3: Grenseverdier for første brett i Lag-A.

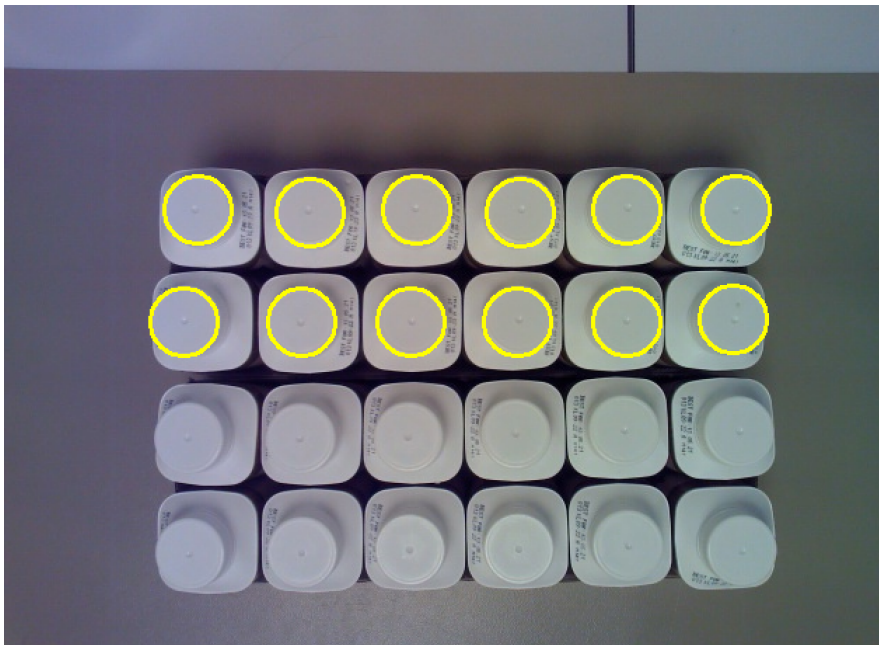
max_x	650
min_x	100
max_y	300
min_y	150

Tabell 3.4: Grenseverdier for neste brett i Lag-A.



Figur 3.17: Første pappbrett som skal plukkes i Lag-A.

Når koordinatene til de de første 12 korkene, se i figur 3.19a, er sendt til roboten skal de 12 korkene i neste pappbrett detekteres. Da blir differansen mellom grenseverdiene i tabell 3.3 og 3.4 lagt inn i listing 3.10. Resultatet av dette vises i figur 3.18 og koordinatene som printes ut vises i figur 3.19b.



Figur 3.18: Neste pappbrett som skal plukkes i Lag-A.

```
for (x, y, r) in circles:
    if min_x < x < (max_x + 100) and (min_y - 150) < y < (max_y-150):
        a = cv2.circle(output, (x, y), r, (0, 255, 255), 2)
for i in circles:
    if (min_y - 150) < i[1] < (max_y-150) and min_x < i[0] < (max_x +
   100):
        b = cv2.circle(output,(i[0],i[1]),2,(0,0,255),3)
        print(i)
```

Listing 3.10: Kode for deteksjon av neste pappbrett i Lag-A.

```
[220 408  24]
[298 406  24]
[374 406  24]
[452 402  24]
[220 326  24]
[140 326  24]
[140 408  24]
[532 322  24]
[534 406  24]
[372 324  24]
[296 326  24]
[450 322  24]
```

(a) Første pappbrett

```
[302 164  24]
[530 164  24]
[452 244  24]
[220 244  24]
[452 164  24]
[136 244  24]
[226 166  24]
[528 242  24]
[146 164  24]
[298 244  24]
[376 166  24]
[374 244  24]
```

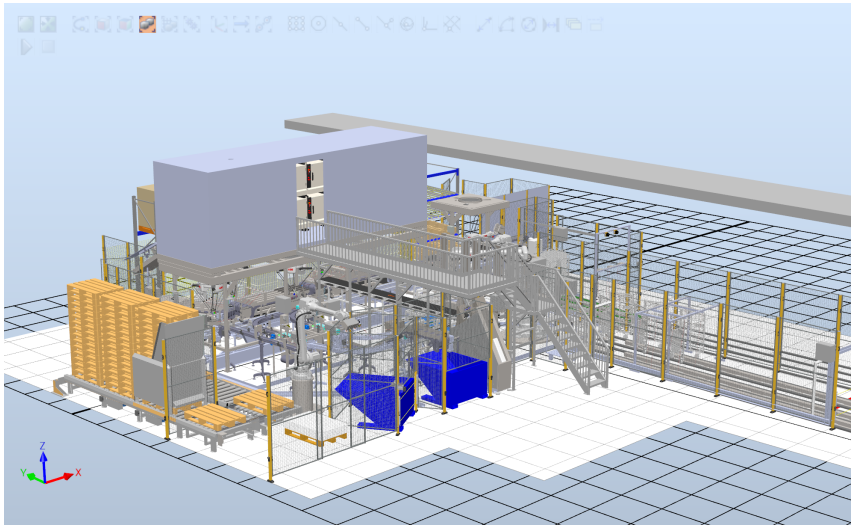
(b) Andre pappbrett

Figur 3.19: Koordinatene til pappbrett i Lag-A.

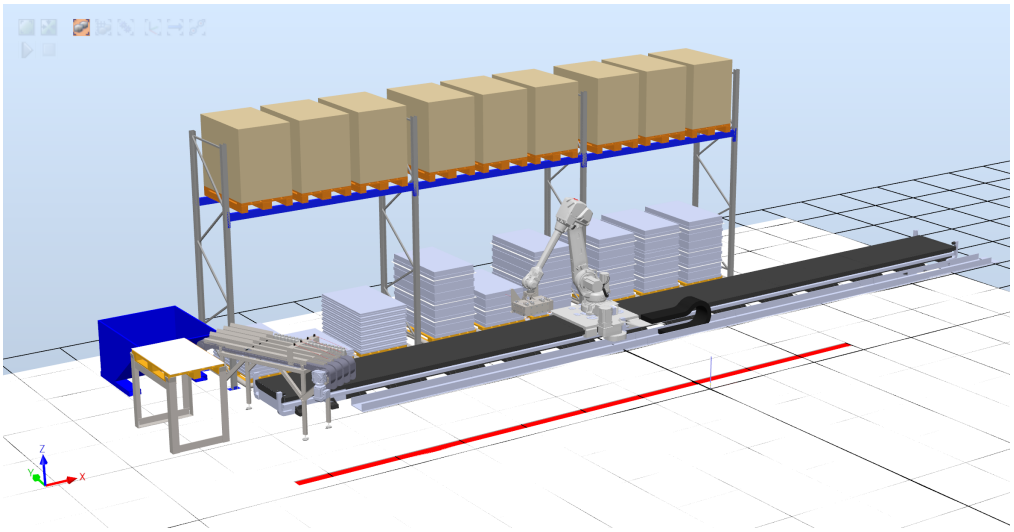
3.3 Simulering i RobotStudio

En god simulering av prosessen kan vise bevegelser og timing slik at resultatet kan vurderes før det bygges. Spesielt ulike layout og alternativer kan vurderes opp mot hverandre for å se hva som gir de beste resultatene.

Derfor ble det laget en simulering av denne delen av prosessen. RobotStudio er et hensiktsmessig verktøy for dette, derfor det ble brukt til å lage miljøet og simuleringen. En Pack- and go fil ble mottatt av RobotNorge av hele Tine-prosjektet som de har lagd, se figur 3.20. Deretter ble de delene som ikke var en del av oppgaven fjernet for en bedre oversikt, se figur 3.21.



Figur 3.20: Stasjonen som viser Tine-prosjektet i sin helhet. Stasjonen er laget av Robot-Norge for Tine SA.



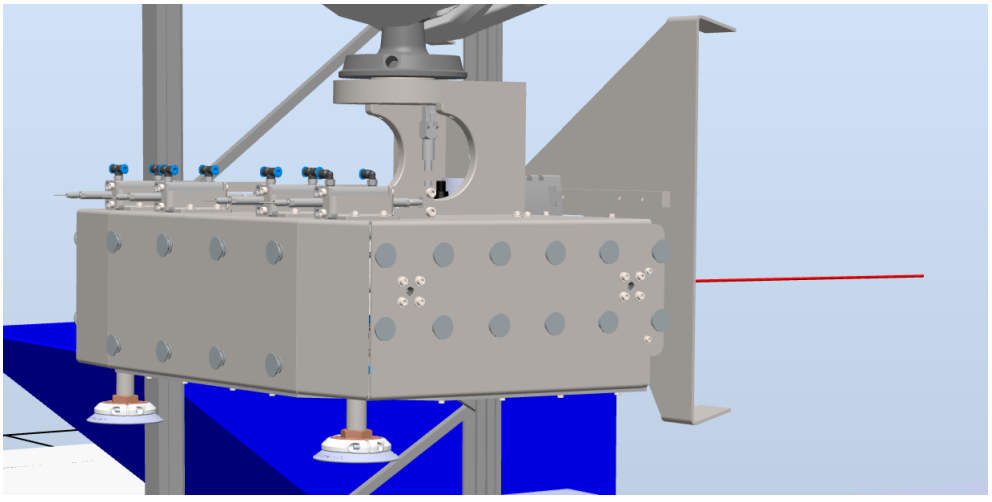
Figur 3.21: Delen av Tine-prosjektet som representere denne oppgaven, hvor resten av stasjonen er blitt fjernet. Utviklet av RobotNorge.

3.3.1 Transportbånd

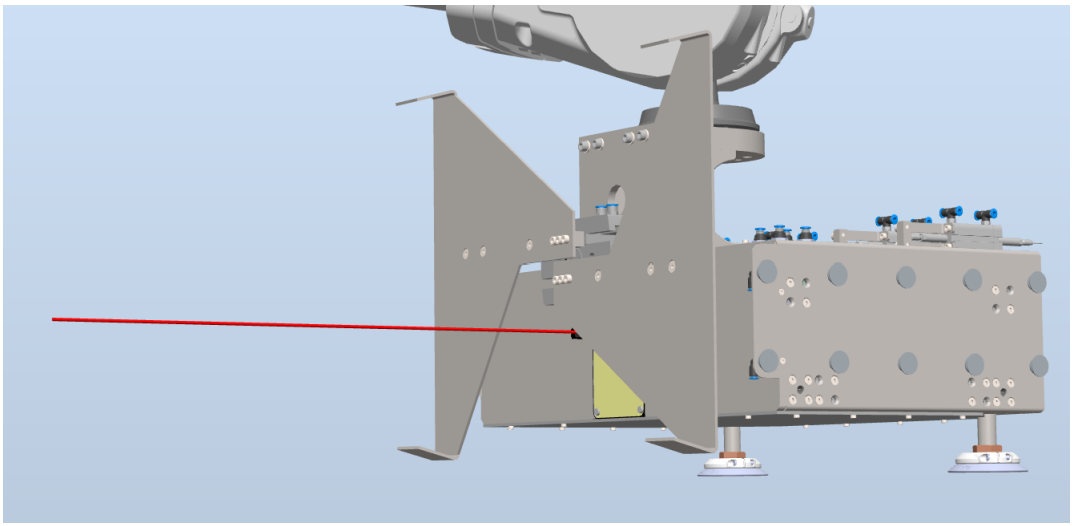
Roboten som ble brukt til oppgaven var av typen IRB4600_40_255. Denne settes på et bånd slik at roboten får syv akser og kan flytte seg langs pallene med produkter på. Det 12 meter lange transportbåndet er av typen IRBT4004. Dette settes opp ved hjelp av guiden fra ABB, se vedlegg 6.1. Dette blir gjort for at transportbåndet skal kunne programmeres og simuleres slik som resten av roboten. På figur 3.21 går transportbåndet forbi pallene som er satt opp og grunnen til dette er at det vil ligge andre produkter som er ikke er en del av oppgaven.

3.3.2 Griper

Griperen som står på roboten er designet at RobotNorge og i denne delen brukes sugekopper til å frakte produktene. På den ene siden av griperen er det lagt inn kamera og avstandssensor slik at en får nøyaktige avstandsverdien hver gang. Det er også et feste på den samme siden, men dette er til produkter som ikke er en del av oppgaven. På de tre andre sidene av griperen er det sugekopper. Disse er plassert etter hvilket produkt de skal tilpasse og det er dermed tre forskjellige muligheter til å plukke produktene. Det skilles mellom produkter som ligger i melkekartonger med rund kork på toppen og forskjellige yoghurter. Figur 2.2 viser hvordan melkebrettene ser ut fra siden. Se vedlegg 6.3 for resten av produktene. I figur 3.22 på siden med 12 sugekopper kan en se at mellom de ytterste sugekoppene på begge sidene er det noe på griperen. Dette er sylindere som går gjennom pappbrettet produktene står på slik at de er festet og da kan frigjøres i konteineren. På figur 3.23 er kamera plasert bak det gule glasset på griperen og den røde streken er avstandssensoren. De store platene på samme side som kameraet er plasert er et feste til en del som er utenfor oppgaven og blir derfor ikke brukt her.

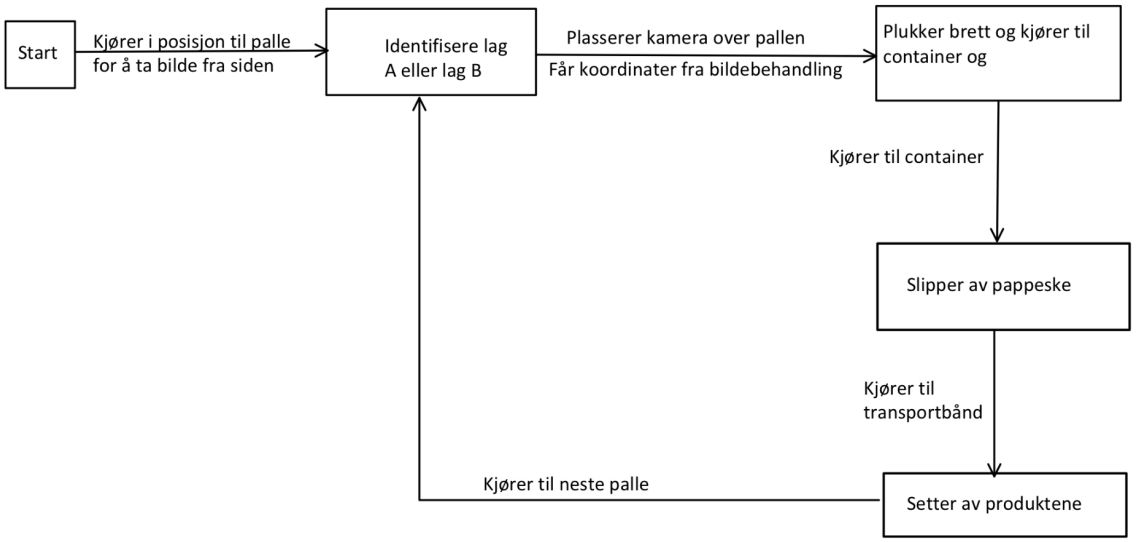


Figur 3.22: Viser sidene til griperen med 8 sugekopper og siden med 12 sugekopper til melkekartongene. Utviklet av RobotNorge.



Figur 3.23: Viser sidene til griperen med 10 sugekopper og den andre siden med kamera og avstadnssensor. Utviklet av RobotNorge.

3.3.3 Simulering



Figur 3.24: Tilstandsdiagram som viser gangen i simuleringen.

Først plasseres pallene med brett med produkter i de angitte plassene av operatører og plastikken rundt pallene fjernes for hånd. Så vil roboten kjøre slik som vist i tilstandsdiagrammet 3.24 og gjenta dette på hver palle. Det vil si at den vil ta et brett av palle nummer 1 og så gå videre til palle nummer 2 og så videre. Roboten vil da gå helt bort til palle nummer ni og plukke et brett av den. Deretter vil den begynne på palle nummer en igjen og gjenta prosessen. Se figur 3.24 for en oversikt over hva roboten skal gjøre. Dette gjentas for hver palle.

3.3.4 Kommunikasjon mellom Python og RobotStudio

Tilstrekkelig rask og sikker kommunikasjon mellom Python og RobotStudio er nødvendig for at systemet skal fungere. For å få programmene til å kommunisere med hverandre brukes den fysiske IP-adressen til roboten for å koble dem sammen.

```
robot = RWS.RWS("")

robot.set_rapid_variable("action", action)
robot.set_rapid_variable("image_processed", image_processed)
wait_for_rapid(robot, "ready_flag")

def wait_for_rapid (self, var='ready_flag'):


    while self.get_rapid_variable(var) == "FALSE" and self.is_running()
   :

        time.sleep(0.1)
        self.set_rapid_variable(var, "FALSE")
```

Listing 3.11: Kommunikasjon mellom Python og RobotStudio.

Koden bruker *RWS.RWS("IP-adresse")* for å få kontakt med den fysiske roboten og dette gikk ikke med simulering. Dermed ble det ikke opprettet kommunikasjon mellom bildebehandlingsdelen og simuleringen i RobotStudio.

Resten av koden går ut på å sjekke hvilket action som skal brukes, også sjekke om bilde er prosessert. Dersom bildet er prosessert og RAPID er ferdig vil det sende et *Ready_flag* til Python for å vite at prosessen er fullført.

3.3.5 Simulering

For å lage simuleringen brukes RobotStudio og RAPID. For å simulere bevegelsen roboten vil flytte seg ble MoveL og robtarget brukt til å lage en path roboten følger.

```
    PROC main()
        Path_20;
    ENDPROC
    PROC Path_20()
        MoveL Target_110,v1000,z100,tool0\WObj:=wobj0;
        MoveL Target_120,v1000,z100,tool0\WObj:=wobj0;
        MoveL Target_190,v1000,z100,tool0\WObj:=wobj0;
        MoveL Target_200,v500,z100,tool0\WObj:=wobj0;
        MoveL Target_210,v500,z100,tool0\WObj:=wobj0;
        WaitTime 5;
        MoveL Target_220,v100,z100,tool0\WObj:=wobj0;
        MoveL Target_250,v1000,z100,tool0\WObj:=wobj0;
        WaitTime 5;
        MoveL Target_380,v500,z100,tool0\WObj:=wobj0;
        WaitTime 5;
```

```
16          MoveL Target_390,v500,z100,tool0\WObj:=wobj0;
17          MoveL Target_400,v500,z100,tool0\WObj:=wobj0;
18          WaitTime 5;
19          MoveL Target_410,v500,z100,tool0\WObj:=wobj0;
20          WaitTime 5;
21          MoveL Target_420,v500,z100,tool0\WObj:=wobj0;
22          MoveL Target_430,v500,z100,tool0\WObj:=wobj0;
23          WaitTime 5;
24          MoveL Target_550,v1000,z100,tool0\WObj:=wobj0;
25          MoveL Target_560,v1000,z100,tool0\WObj:=wobj0;
26          WaitTime 5;
27          MoveL Target_450,v100,z100,tool0\WObj:=wobj0;
28          WaitTime 2;
29          MoveL Target_580,v1000,z100,tool0\WObj:=wobj0;
30          MoveL Target_460,v1000,z100,tool0\WObj:=wobj0;
31          MoveL Target_470,v1000,z100,tool0\WObj:=wobj0;
32          WaitTime 3;
33          MoveL Target_480,v100,z100,tool0\WObj:=wobj0;
34          WaitTime 3;
35          MoveL Target_490,v1000,z100,tool0\WObj:=wobj0;
36          MoveL Target_500,v500,z100,tool0\WObj:=wobj0;
37      ENDPROC
38  ENDMODULE
```

Listing 3.12: Kode i RAPID for simulering av roboten.

Linje 5 til 9 i koden flytter roboten til posisjon 3 for å ta bilde fra siden på pallen. Deretter venter roboten i 5 sekunder og i praksis er det her bildebehandlingen skjer. Så i linje 11 og 12 flytter roboten kameraet over slik at de kan lokalisere koordinatene til korkene. Bildebehandlingen skjer igjen i ventetiden som står på linje 13. Linje 14 justerer griperen slik at sugekoppene treffer alle korkene og linje 15 er en ventetid for at sugekoppene skal aktiveres. Så kjøres roboten bort til konteineren ved kommandoen i linje 16 og 17 og den løsner pappbrettet ved ventetiden på linje 18. Deretter flytter roboten seg til transportbåndene hvor den setter av melkekartongene. Dette skjer på linje 19 og 20. Så flytter roboten seg til posisjon 4 og gjentar de samme stegene. Se delkapittel 4.1.1 for link til video av simuleringen.

Koden viser simuleringen av hvordan roboten vil kjøre for Lag-A og Lag-B. I tillegg ville det vært kode for å aktivere og deaktivere sugekoppene og lagt inn pauser der roboten venter på klar-flagg fra Python. Dette vises i koden hvordan dette ville sett ut fra koden som ble lagd i ELE610 fra RS-5. Tilsvarende kode ville blitt lagd dersom det ville vært mulighet for å simulere på en virkelig robot.

```
1       PROC main()
2           WHILE TRUE DO
3               wait_for_python;
4               TEST action
5                   CASE 1:
6                       MoveL Offs(target_ow, 0, 0, 200),v500,z10,tGripper\
    WObj:=wobjTableN;
```

```
                    ready_flag := TRUE;
                CASE 2:
                    MoveL Offs(target_cw, 0, 0, 200),v500,z10,tGripper\
   WObj:=wobjTableN;
                    ready_flag := TRUE;
                CASE 3:
                    simple_robtarget := target_K1;
                    getPuck;
                CASE 4:
                    simple_robtarget := target_K0;
                    putPuck;
            ENDTEST
        ENDWHILE
    ENDPROC
```

Listing 3.13: Kode i RAPID for kommunikasjon hvor roboten venter på koordinater fra Python. Dette er fra RS-5.

Koden for å aktivere og deaktivere sugekoppene vil se lignende ut på en fungerende smart component. For å aktivere sugekoppene på de rette tidspunktene ville actions og case brukt slik at f.eks. i case 3 ville sugekoppene blitt aktivert og melkekarongene ville vært festet til griperen.

```
PROC main()
    SetDO doVacuum, 0;
    WaitDI diVacuum, 0;

    WHILE TRUE DO
        PickPack;
    ENDWHILE
ENDPROC
```

Listing 3.14: Kode i RAPID for aktivering av Smartkomponenter.

Kapittel 4

Tester og diskusjon

4.1 Tester

4.1.1 Simulering

Trykk på linken for å se video av simulering, link til video. Simuleringen brukte 99.9 sekunder på to paller. Ettersom dette kun er en simuleringen og det ikke er testet i praksis vil det være vanskelig å bestemme om roboten kunne beveget seg raskere. Eventuelt om roboten burde kjørt saktere ved noen plasser under simulering. De mest nøyaktige bevegelsene, som plukking av produkter, utføres saktere enn for eksempel når roboten flytter seg langs transportbåndet. Dette er for å få økt nøyaktighet for å unngå feil.

4.1.2 Bildebehandling

Vedlegg 6.5 viser koden for HSV filteringen. Denne ble brukt for å finne de rette verdiene som skulle inn i hovedkoden. Hovedkoden ligger i vedlegg 6.4. Den er i stand til å detektere et pappbrett med produkter og deretter sender posisjonen til hver kork i brettet, i form av koordinater, til roboten. Selv om koden fra bildebehandlingen fungerte som ønsket ble det ikke gjennomført en samkjøring med robot. Av den grunn vil resultatet fra en praktisk test være usikkert men forhåpentligvis vil avvik ikke forekomme eller i det minste være lite.

4.2 Diskusjon

I dette delkapittelet vil det diskuteres ulike utfordringer som ble oppdaget underveis i prosjektet og eventuelle løsninger til utfordringene.

4.2.1 Paller

Som nevnt i innledningen av kapittel 3 var mangelen på tilgang til en hel palle som bildebehandlingen kunne testes på en utfordring. Dermed ble bildebehandlingen utført etter beste evne ved bruk av to brett med produkter. Brettene ble lagt slik at det ble utført tester både for Lag-A og Lag-B, se figur 2.3 for hvordan de forskjellige lagene ser ut.

4.2.2 Kameraet

Kameraet er enkelt å sette opp og lett å få i bruk. Utfordringer som ble oppdaget på veien var at kamera er uforutsigbart ved forskjellige lyssettinger. Dermed ble filterene som ble brukt i bildefangsten ubrukelige til tider og det som tidligere hadde gitt gode resultater gidde nå dårlige resultater. Altså var det veldig uforutsigbart og med denne erfaringen ville det dermed vært klokt å valgt et annet kamera, som for eksempel et smart-kamera fra Omron slik som RobotNorge bruker i sin løsning av problemstillingen.

4.2.3 Smart Komponent

Smart komponent er en god måte å gi verktøy egendefinerte egenskaper som for eksempel at sugekopper fungere. Dessverre har dette ikke vært mulig på griperen som ble sendt fra RobotNorge. Ifølge bruksanvisningen som ligger i vedlegg 6.2 er delen som blir brukt til å lage en smart komponent lagt inn under definisjonen Mechanics mens griperen fra RobotNorge går under Components. Da blir det ikke helt likt som bruksanvisninger viser og den fungerer ikke.

4.2.4 Kommunikasjon

Kommunikasjonen mellom Python og RAPID har blitt vist i kapittelet om konstruksjonen av oppgaven, se kapittel 3. Her refereres det til RS-5 fra ELE610 for hvordan man oppnår kommunikasjon mellom den fysiske roboten ved hjelp av IP-adresse og det ville vært tilsvarende for denne oppgaven. Da det ikke var en mulighet å få testet systemet i praksis var det vanskelig å gjennomføre og det var ikke tid og anledning til å prøve å opprette kommunikasjon mellom Python og simuleringen i RobotStudio.

4.2.5 Simulering

Simuleringen viser hvordan roboten vil bevege seg, men da uten kobling opp til bildebehandlingen og fungerende sugekopper. Referer til delkapittelene om tilsvarende deler for hva som gikk galt og hvorfor simulering da ikke kunne utføres som ønsket. Se delkapittel 4.2.3 og 4.2.4 for mer konkret informasjon om hva som var årsaken til at simuleringen ikke ble som ønsket.

Kapittel 5

Konklusjon

Depalletering av produkter ved hjelp av robot har vært en utfordrende og givende oppgave.

Detektering av pappbrett ved hjelp av et enkelt kamera og bruk av OpenCv i Python viste seg å være en større utfordring enn forventet. En av de største utfordringene var å kunne lage en kode som fungerte hver gang under ulike lysforhold. Underveis ble det funnet ut at trackbar, hvor en kan endre på verdiene, var den beste løsningen slik at innstillingene kan endres under forskjellige lysforhold. Utfordringen ved å lokalisere et pappbrett var også større enn først antatt og det gikk en del tid til å gruble over forskjellige løsninger til utfordringen. Flere tips fra forskjellige forum på internettet var med på å løse arbeidet med å få koden til å lokalisere brettene slik at det fungerte meget bra. Bildene fra delkapittel 6.4 er tatt på forskjellige dager, og den samme koden fungerte bra under mindre forskjeller i lysforholdene.

Det ble mindre utfordringer med lokalisering av korkene da blant annet problemet med lysforholdene var løst. Det viste seg også å være enklere å lage kontur rundt korkene sammenlignet med pappbrettene. På grunn av at roboten vil ha fast avstand til korkene hver gang den tar bilde over korkene ble radiusen brukt for å eliminere så mye støy som mulig. Deretter ble koordinatene til korkene funnet slik at disse kunne sendes til roboten for å vite den nøyaktige plasseringen til korkene.

Simuleringen til systemet ble utført i RobotStudio og griperen til roboten ble tilsendt fra RobotNorge. Da griperen skulle defineres som smart komponent ville den ikke fungere som ønsket. Dermed ble simuleringen kjørt uten at sugekoppene ble aktivert, men mønsteret roboten kjører vil være identisk i realiteten.

Problemet med å ikke ha en fysisk robot som står på et transportbånd gjorde at kommunikasjonen mellom roboten og Python ikke ble opprettet som ønsket. For å få forbindelse mellom dem måtte robotens IP-adresse bli tatt i bruk.

Bibliografi

[1] Andreas Kverneland. (06-2018). *RobotNorge: Optimalisere og robotisere pakkelinje [Masteroppgave, Universitet i Stavanger]. Uis Brage.* URL: https://uis.brage.unit.no/uis-xmlui/handle/11250/2564544.

[2] *About*. URL: https://opencv.org/about/. (accessed: 11.05.2021).

[3] Tom Allen. *What is machine vision? Everything you need to know*. URL: https://aijourn.com/what-is-machine-vision-everything-you-need-to-know/. (accessed: 19.05.2021).

[4] *Color - Color models and color spaces*. URL: https://programmingdesignsystems.com/color/color-models-and-color-spaces/index.html. (accessed: 09.04.2021).

[5] *Datei:HSV cylinder.jpg. [Bilde]. Wikipedia*. URL: https://de.wikipedia.org/wiki/Datei:HSV_cylinder.jpg.

[6] Prof. Dr. Amarendra Bhushan Dhiraj. *Countries with the most industrial robots per 10,000 employees, 2018 report*. URL: https://ceoworld.biz/2018/03/14/countries-with-the-most-industrial-robots-per-10000-employees-2018-report/. (accessed: 12.04.2021).

[7] Eiliv Elvebakk. *Robotisering, en nødvendighet for å utvikle norsk industri*. URL: https://blogg.triple-s.no/robotisering-en-n%C3%B8dvendighet-for-%C3%A5-utvikle-norsk-industri. (accessed: 12.04.2021).

[8] Jan Tommy Gravdahl. *Automatisering*. URL: https://snl.no/automatisering. (accessed: 14.04.2021).

[9] *Helautomatiserer TINE Skolelyst*. URL: https://robotnorge.no/helautomatisert-pakkelinje-til-tine-skolelyst/. (accessed: 20.05.2021).

[10] Nordea Markets. *Fem grunner til å automatisere arbeidsprosessene*. URL: https://insights.nordea.com/no/naeringsliv/fem-grunner-til-a-automatisere-arbeidsprosessene/. (accessed: 12.04.2021).

[11] Marie Mathisen. *5 fordeler med robotisering*. URL: https://www.visma.no/blogg/5-fordeler-med-robotisering/. (accessed: 27.04.2021).

[12] *Object Detection Guide*. URL: https://www.fritz.ai/object-detection/. (accessed: 21.05.2021).

[13] *Om TINE*. URL: https://www.tine.no/om-tine. (accessed: 14.04.2021).

[14] Universitet i Oslo. *Farger (2011, 4. februar)*. URL: https://www.mn.uio.no/ibv/tjenester/kunnskap/plantefys/leksikon/f/farger.html.

[15] Universitet i Oslo. *TEK5030 – Maskinsyn (2021, 25. mai)*. URL: https://www.uio.no/studier/emner/matnat/its/TEK5030/.

[16] *Robotic Palletizing*. URL: https://www.mhi.org/solutions-community/solutions-guide/robotics. (accessed: 20.05.2021).

[17] *RobotNorge AS, Om oss og vår historie*. URL: https://robotnorge.no/om-oss/. (accessed: 14.04.2021).

[18] Adrian Rosebrock. *Detecting Circles in Images using OpenCV and Hough Circles*. URL: https://www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles/. (accessed: 23.03.2021).

[19] Om Singh. *Trackbar in OpenCV Python*. URL: https://blog.electroica.com/trackbar-in-opencv-python/. (accessed: 10.05.2021).

[20] Sourabh Sinha. *Filter Color with OpenCV*. URL: https://www.geeksforgeeks.org/filter-color-with-opencv/. (accessed: 05.04.2021).

[21] Karl Skretting. *ELE610 - Robot technology - ABB robot assignment 1. 2021.*

[22] Karl Skretting. *ELE610 - Robot technology - ABB robot assignment 5. 2021.*

[23] *TDT4265 - Datasyn og dyp læring*. URL: https://grades.no/course/TDT4265. (accessed: 19.05.2021).

[24] *Vision sensors & intelligent cameras*. URL: https://www.vision-doctor.com/en/smart-cameras.html. (accessed: 16.05.2021).

[25] *What is the difference between a framework and a library?* URL: https://stackoverflow.com/questions/148747/what-is-the-difference-between-a-framework-and-a-library. (accessed: 21.05.2021).

Kapittel 6

Vedlegg

6.1 External axis

1. External Axes

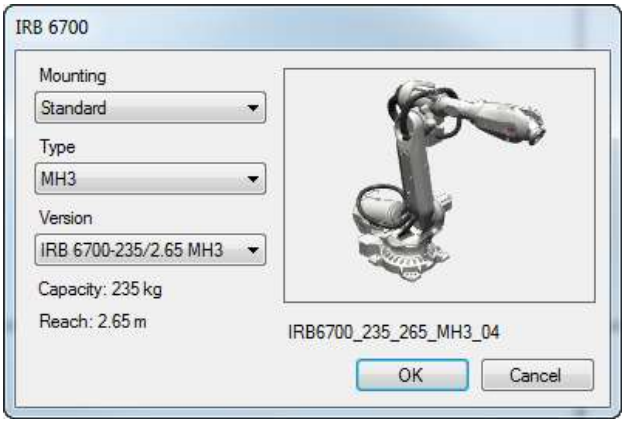
1.1. Setting up a Track Simulation

Overview

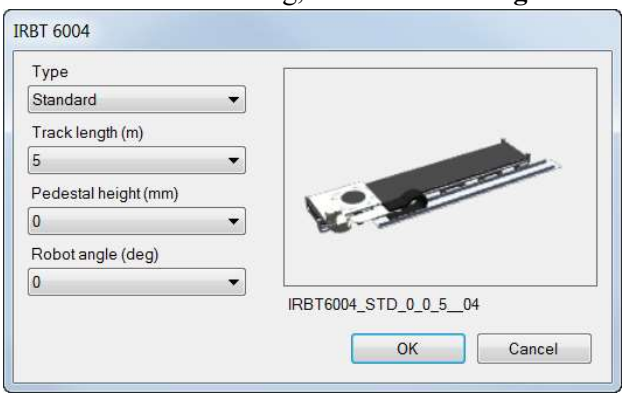
In this exercise we will create a simulation that includes a robot on a track. We will use the system from layout function to create this system.

Layout

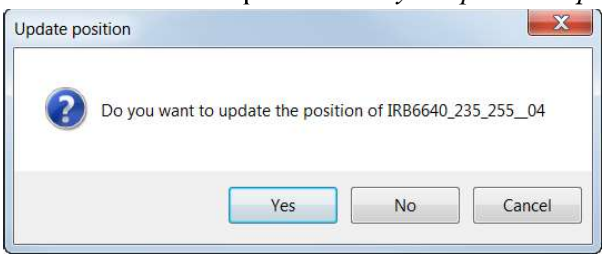
1. On the **File** menu select **Empty Station** and click **Create**.
2. On the **Home** tab click the **ABB Library** button, in the gallery select **IRB 6700**.
3. In the **IRB 6700** dialog, in the **Type** box select **MH3** and in the **Version** box select the **IRB6700-235/2.65**. Click **OK**.



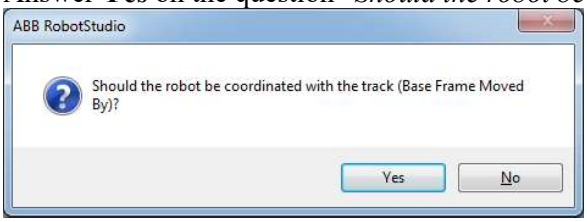
4. Again, on the **Home** tab click the **ABB Library** button, select **IRBT 6004** from the **Track** list.
5. In the **IRBT 6004** dialog, in the **Travel length** combo box, select **5** and click **OK**.



6. In the **Layout** browser drag *IRB6700_235_265_MH3_04* and drop it on the **IRBT 6004**.
7. Answer **Yes** on the question "*Do you update the position of...?*"



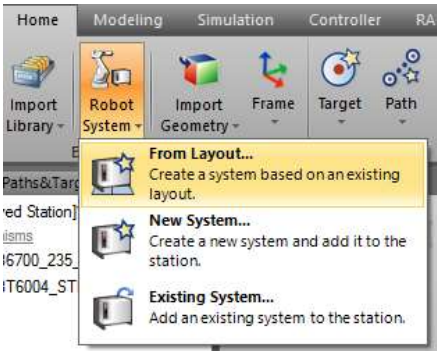
8. Answer **Yes** on the question "*Should the robot be coordinated with the track?*".



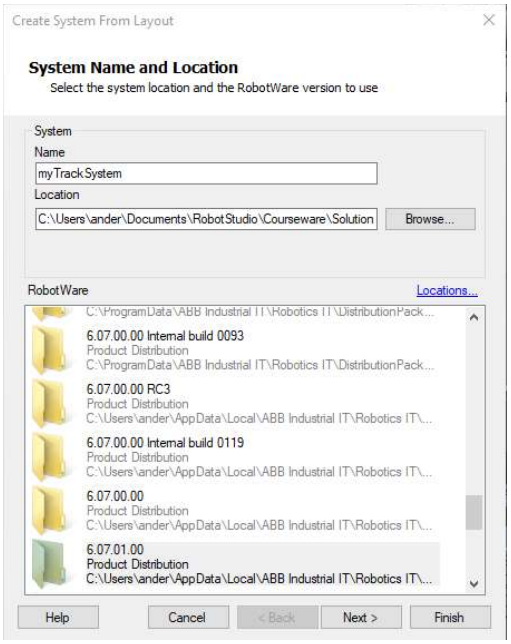
Create System from Layout

In order to create a **System from Layout** RobotStudio requires the configuration files for both the robot and external axis. With RW6.x these configuration files are loaded automatically when RobotStudio is installed. For RW 5.1x versions of RW you need to install the **Track Motion** mediapool located in **Additional Options** folder in the RobotWare download. (unless it was installed previously)

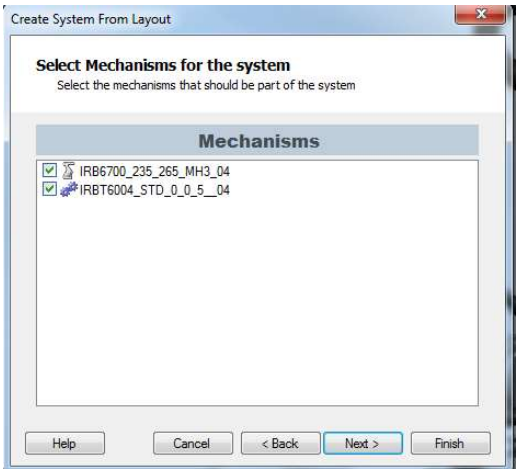
1. On the **Home** tab click the **Robot System** button and select **From Layout**.



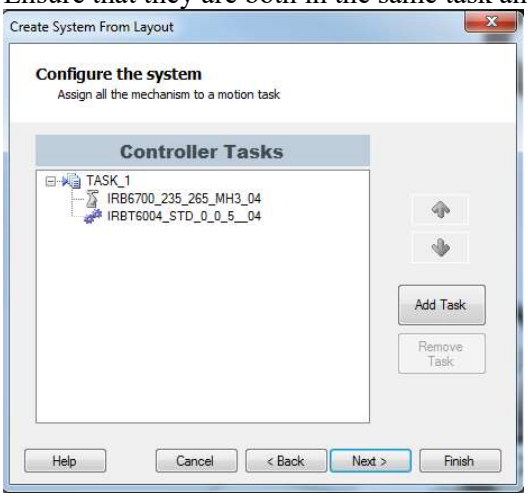
2. In the **Name** field enter *myTrackSystem.* For the location browse to and put it in a new folder *courseware/Solutions/Module_8/myTrackSolution*. Select the verision of RW you wish to use and click the **Next** button.



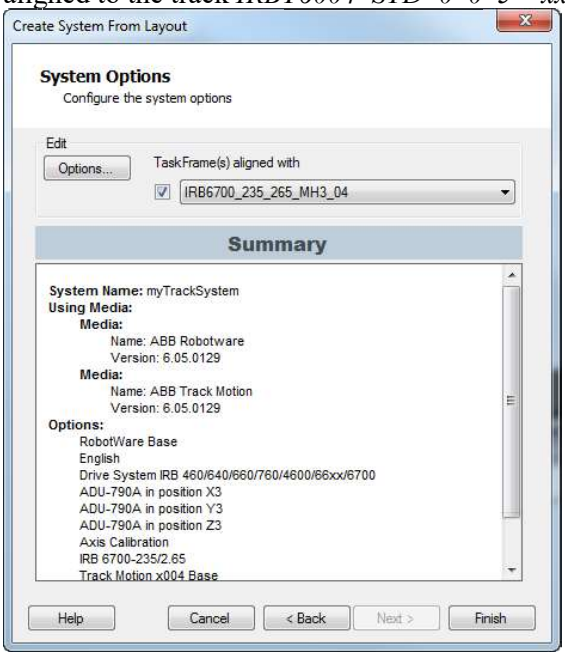
3. Ensure that both mechanisms are selected and click the **Next** button



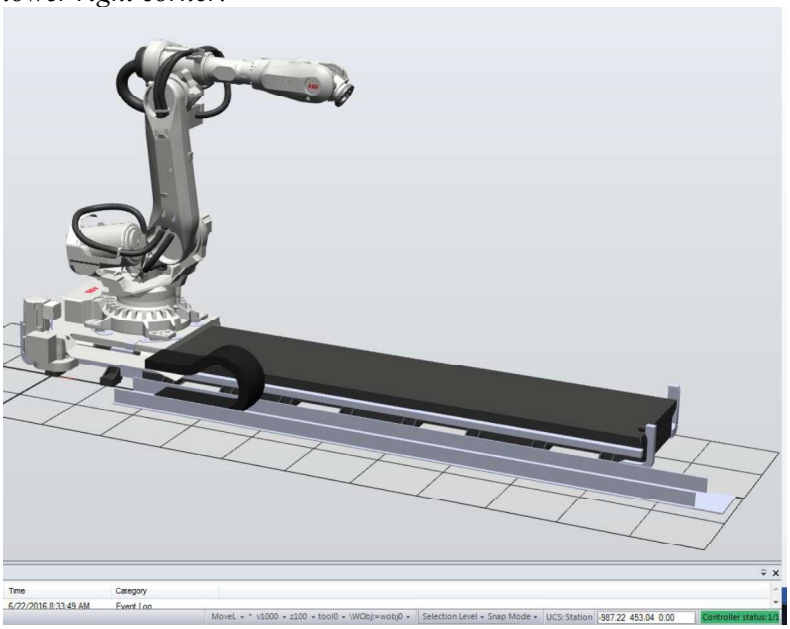
4. Ensure that they are both in the same task and click the **Next** button.



In **System Options** you can add other options to your system. With the taskframe aligned to the track *IRBT6004_STD_0_0_5__xx* (check box checked), click **Finish**.



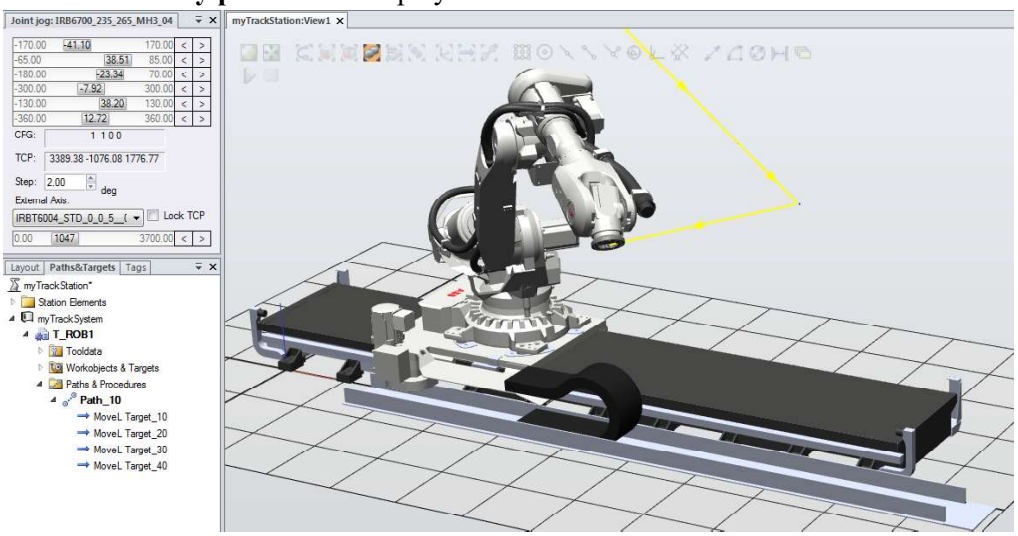
The system starts and when it is fully started you will have a green indication in the lower right corner.



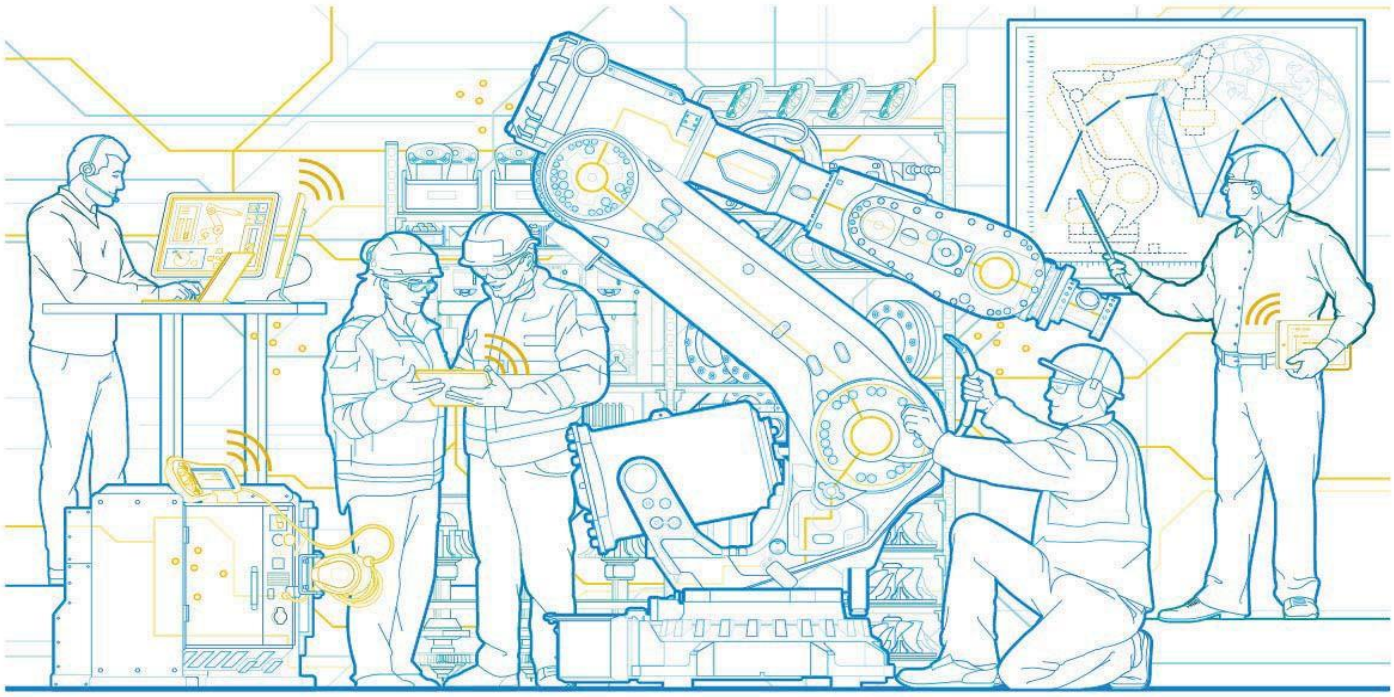
5. **Save** the station at *courseware\solutions\Module_8\MyTrackSolution\myTrackStation*.

Challenge

Create an empty path then use the **Jog Joint** and the **Teach Instruction** functions to create a path of motions that use the coordinated track. **Synchronize to Rapid,** set the path as **simulation entry point** and then play the simulation.



6.2 Smart Components



RobotStudio^{tm} 6.08

SmartComponents

The information in this manual is subject to change without notice and should not be construed as a commitment by ABB. ABB assumes no responsibility for any errors that may appear in this manual.
Except as may be expressly stated anywhere in this manual, nothing herein shall be construed as any kind of guarantee or warranty by ABB for losses, damages to persons or property, fitness for a specific purpose or the like.
In no event shall ABB be liable for incidental or consequential damages arising from use of this manual and products described herein.

This manual and parts thereof must not be reproduced or copied without ABB's written permission, and contents thereof must not be imparted to a third party nor be used for any unauthorized purpose.
Contravention will be prosecuted.
Additional copies of this manual may be obtained from ABB at its
then current charge.

© Copyright 2017 ABB All right reserved.
ABB AB
Robotics Products
SE-721 68
Västerås Sweden

2017-05-08 ABB

1. Smart Components

Overview

Smart Components give RobotStudio users a series of building blocks or tools which enable components within a station to have more complex behavior. Some examples are gipper motion, objects moving on conveyors, logic etc. Once created these objects can be saved as library files for use in future stations and simulations.

1.1. Create Smart Component (SC) - InFeeder

Overview

In this exercise we will learn how to create a Smart Component that simulates an in feeder. It will create dynamic objects which it will move in a straight line until they arrive at a picking position. When an object is removed from the picking position a new object will automatically be created.

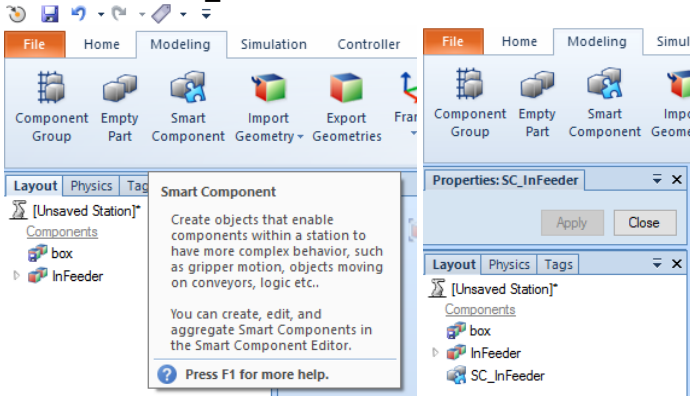
Preparing the station

We will start with an empty station and import the geometry of the conveyor, and the part which should be generated by the SC.

1. Import the geometry ....*Geometry\InFeeder.sat*
2. Import the library ...*Libraries\box.rslib*

Creating the Smart Component

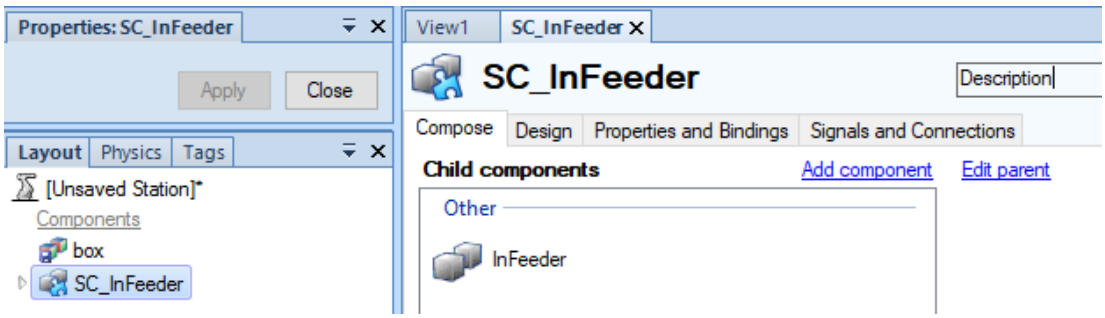
1. On the **Modeling** tab click the **Smart Component** button.
2. Name the SC **SC_InFeeder**



Adding a part to the SC

Next we will add the part **InFeeder** to our SC as a **Child component**.

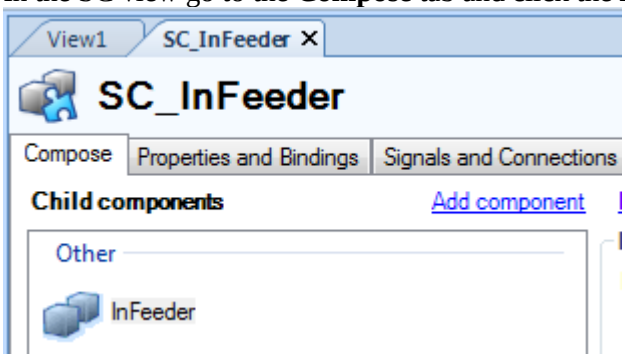
1. In the **Layout** browser drag the part **InFeeder** and drop it on **SC_InFeeder**.



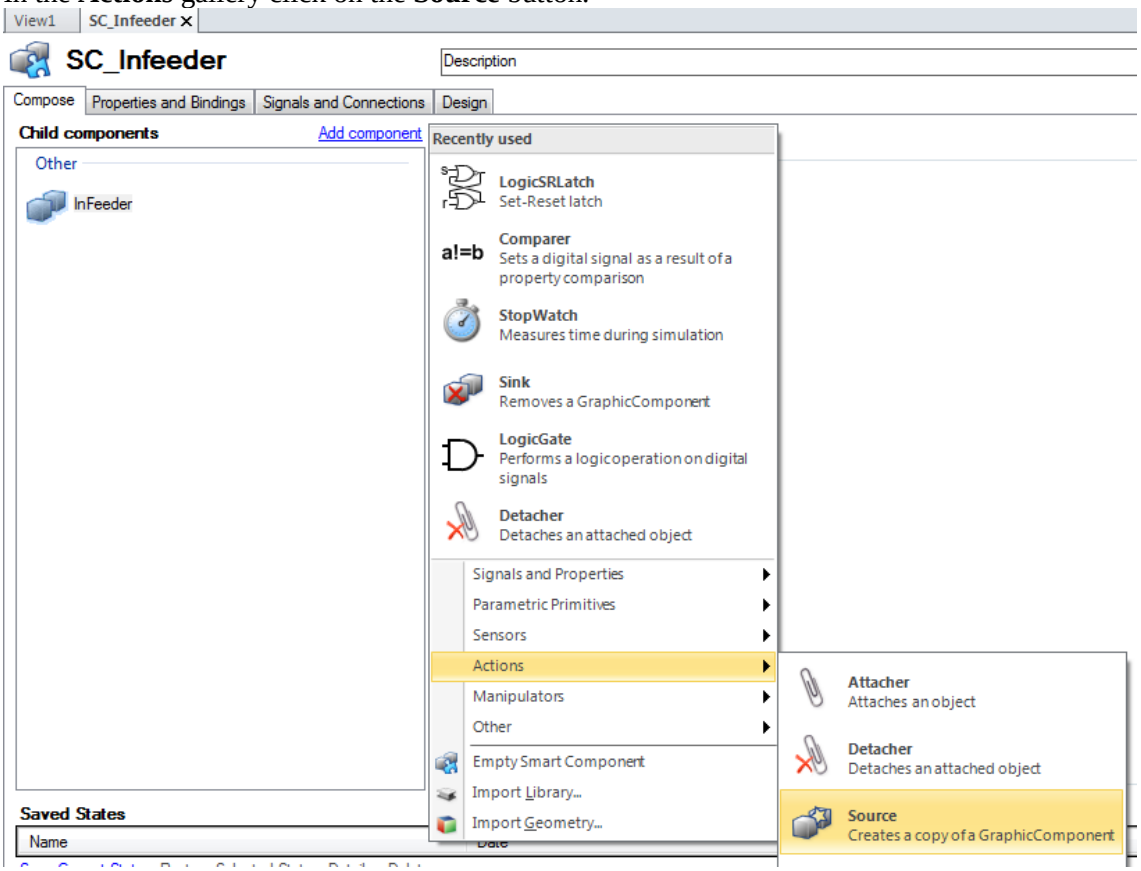
Adding base components to our SC

A Smart Component consists of one or more **base components**. A **base component** is a building block with predefined behaviors and properties.

1. In the SC view go to the **Compose** tab and click the **Add component** button.

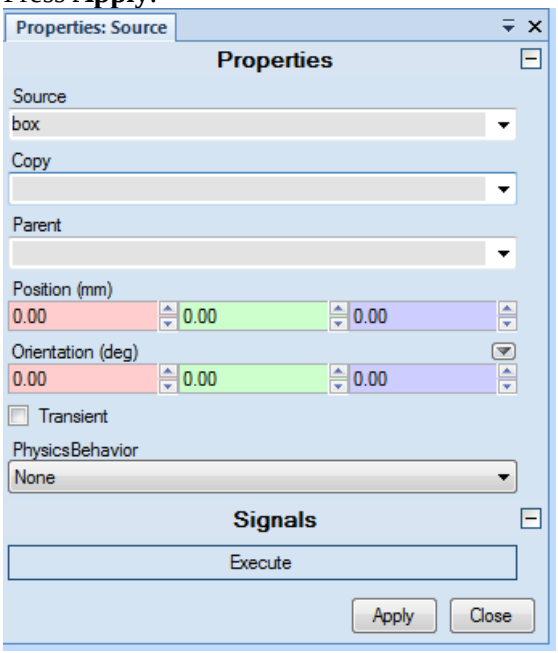


2. In the **Actions** gallery click on the **Source** button.

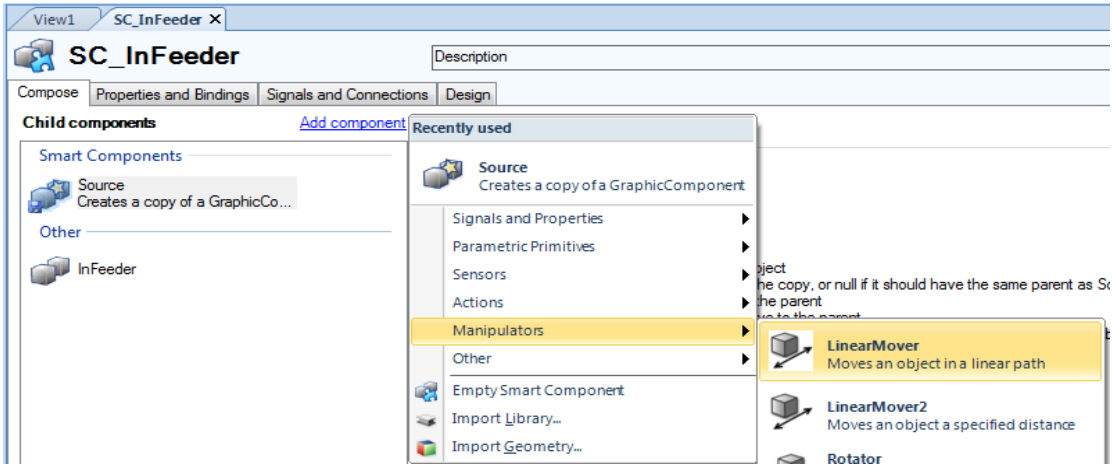


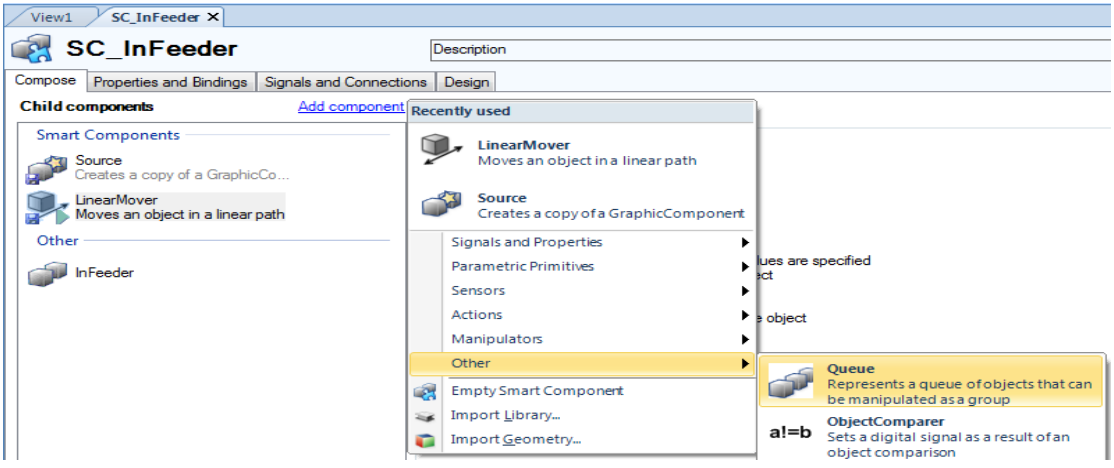
3. Right click on the **Source** and select **Properties** in the context menu (if the properties window not already is opened).
4. In the Properties window select the **box** in the **Source** drop down box.
   NOTE: If the Transient box is checked the part (in this case the box) will automatically be deleted when the simulation is stopped. Leave the box unchecked for this exercise.

5. Press **Apply**.



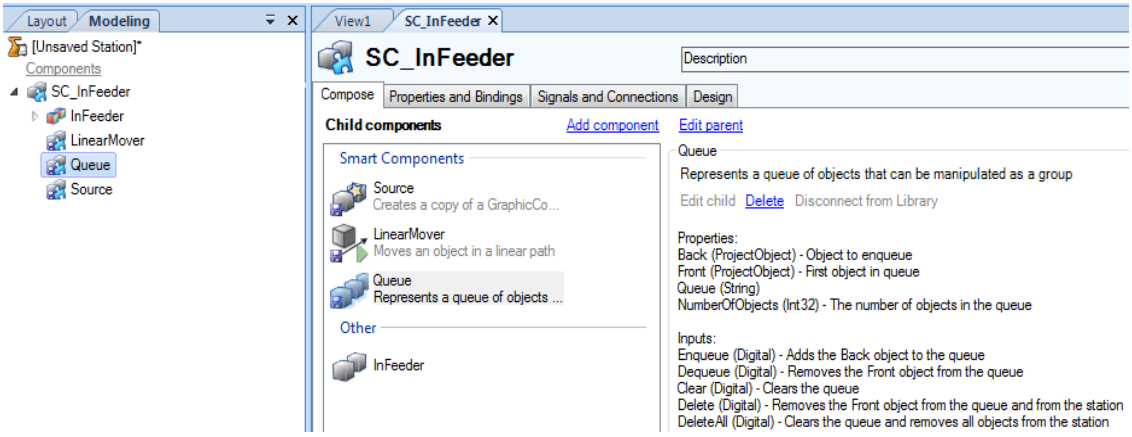
6. Add the base component **LinearMover** from the **Manipulators** gallery, and the base component **Queue** from the **Other** gallery.



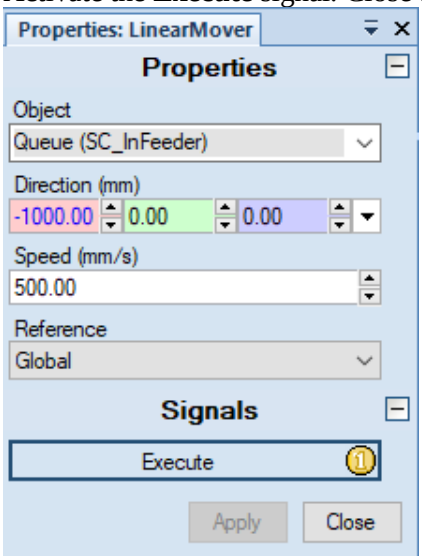


Note: As additional components are added they will populate in the recently used section.

7. The SmartComponent should now have the following features present.



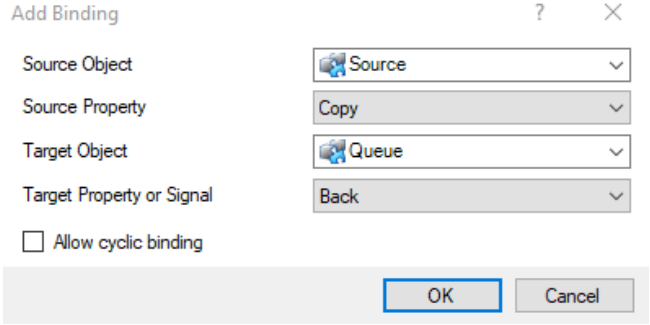
8. Open the Properties window for the **LinearMover** by right clicking on it in the **Layout** browser. Select the **SC_InFeeder/Queue** as the Object. Change the **Direction** to X= -1000, Y=0, Z=0 (the box will move in minus X direction). Change the **Speed** to 500mm/s. Press **Apply**.
9. Activate the **Execute** signal. Close the Properties window.



Adding Property Bindings

The next step is to add a binding between the base component **Source** and the base component **Queue.** This will define which object will be moved by the conveyor.

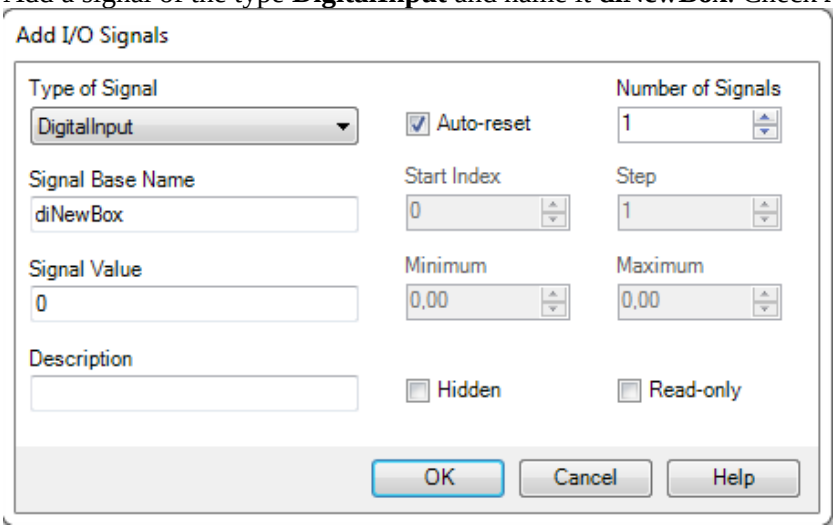
1. In the SC view, go to the **Properties and Bindings** tab and click the **Add Binding** button.
2. Bind the **Copy** of the **Source** to the **Back** of the **Queue** and press **OK**:



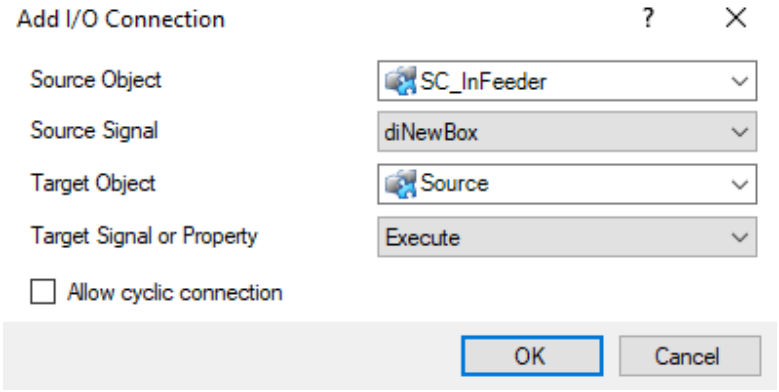
Adding Signals and Connections

In order to define actions in our SC we need to define **I/O Signals** and **I/O Connections**.

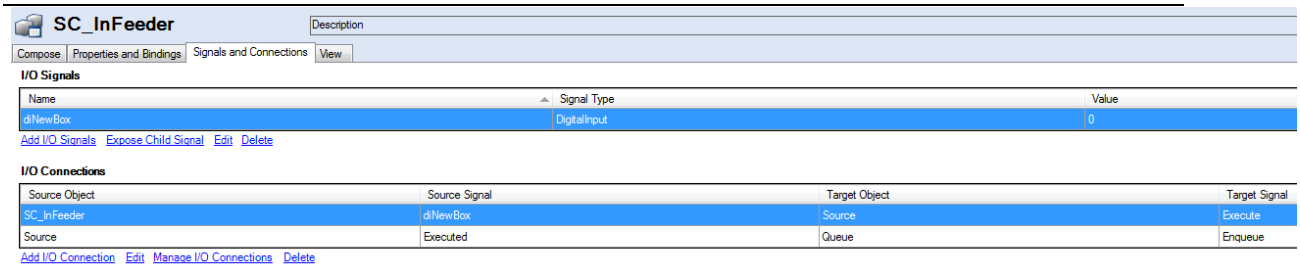
1. In the SC view, go to the **Signals and Connections** tab and click the **Add I/O Signals** button.
2. Add a signal of the type **DigitalInput** and name it **diNewBox**. Check **Auto-reset**.



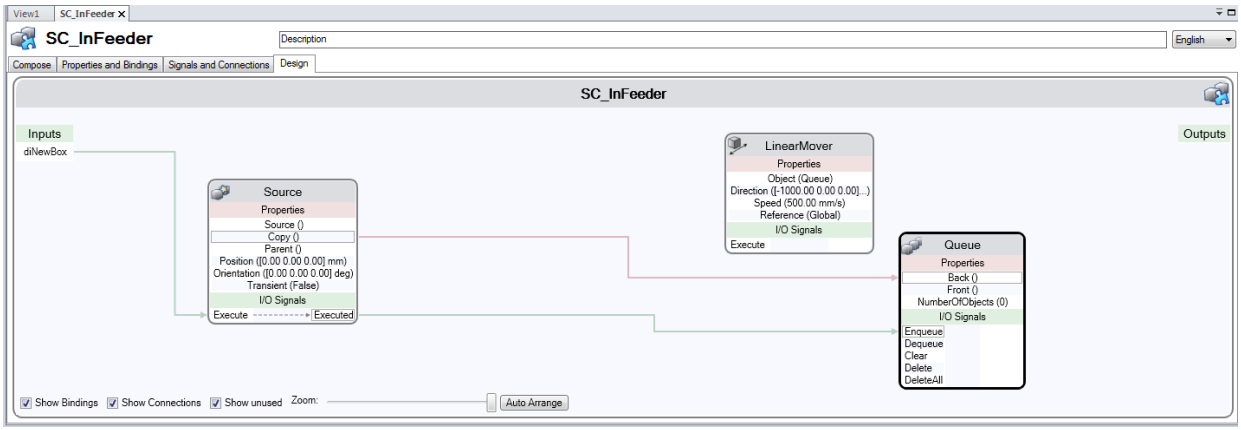
3. In the SC view, go to the **Signals and Connections** tab and click the **Add I/O Connections** button.
4. Set **Source Object** to **SC_InFeeder**, **Source Signal** to **diNewBox**, **Target Object** to **Source** and **Target Signal** to **Execute**. Press **OK**.



5. Add a new **I/O Connection** and set **Source Object** to **Source**, **Source Signal** to **Executed**, **Target Object** to **Queue** and **Target Signal** to **Enqueue**. Press **OK**.



Note: At any point you can also look at the Design view which gives you a graphical representation of the Smart Component. I/O Connections are represented by green lines while bindings are represented by red lines. These can be toggled on and off to simplfy the view if necessary.



6. **Save** the station as *…\my_SC_InFeeder*.

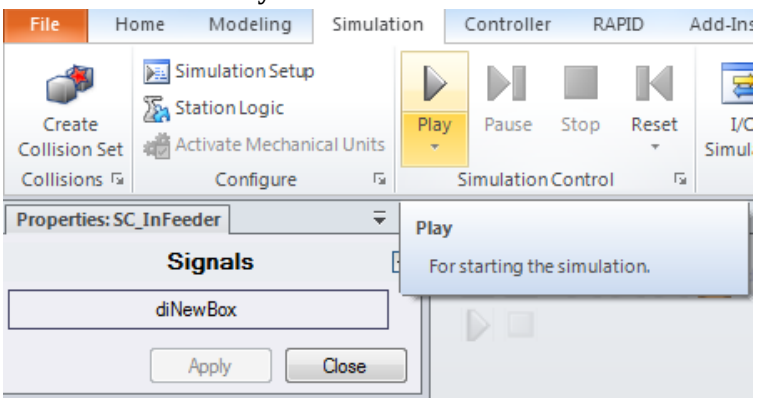
Test running the Smart Component

When you are creating **Smart Components (SC)**, you can minimize the risk of making mistakes by testing it as often as possible. For example, if a sensor is added, you can test the SC by activating the sensor and then moving an object to intersect the beam. The **SensorOut** signal should go high when the beam is intersected.

Before we add a sensor to this SC we will test the **Source** and the **Queue** to ensure it works as expected.

Note: SmartComponents that consume time (e.g. moving objects) are activated by the simulation function within RobotStudio. Therefore a simulation needs to be running in order to test/use a SC.

1. Click on the **my_SC_InFeeder:View1** tab so you can see the conveyor and the box.
2. Open the property window for **SC_InFeeder**.
3. Press **Simulation Play**.

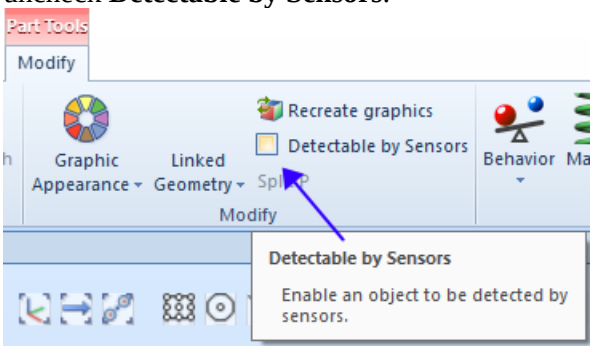


4. Click on **diNewBox** a couple of times and verify that a new box is created and starts moving for each click.
5. Stop the simulation and delete all boxes generated during the simulation.

Adding a Plane Sensor to our Smart Component

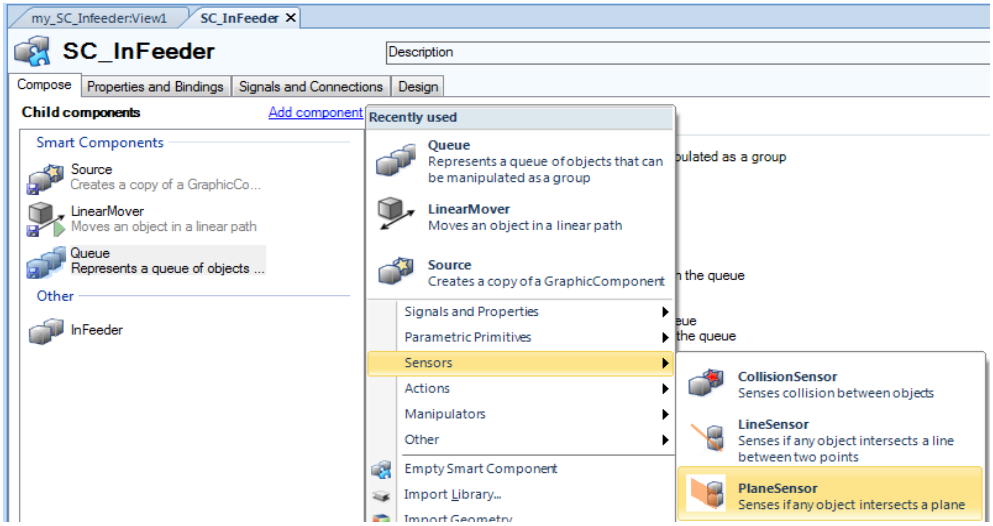
Next we will add a sensor at the end of the conveyor which will stop the box when it hits the sensor beam. We will also add an I/O signal which will remain high as long as a box is in contact with the sensor. Finally we will add an action which generates a new box when a box is removed from the sensor. (We will have the robot pick up the box.)

1. Select the *InFeeder* in the **Layout** browser, then click on **Part Tools-Modify**, and then uncheck **Detectable by Sensors**.

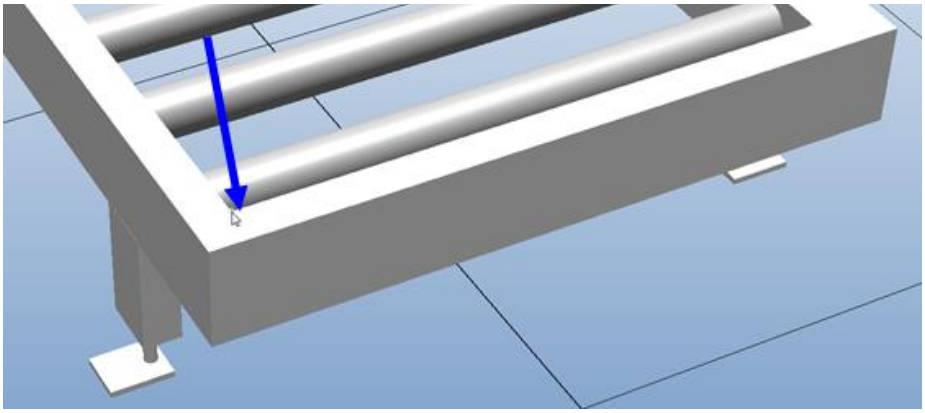


2. This is done to prevent the **PlaneSensor** from detecting the *InFeeder* geometry. It will also prevent the robot's tool from detecting and attaching to the *InFeeder*.

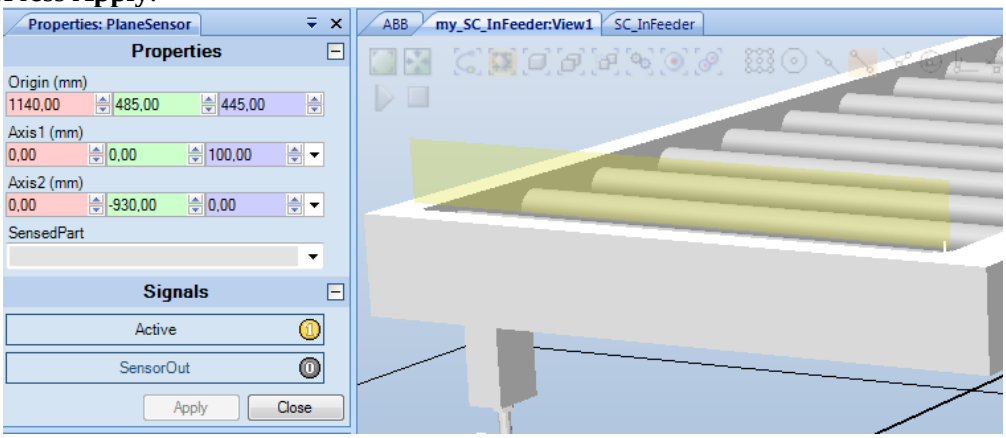
3. In the **Sensors** gallery click on the **PlaneSensor** button.



4. Place the cursor in the **Origin** field in the property window for the **PlaneSensor**.
5. Set up your **selection level** and **snap mode** to enable you to select the corner of the conveyor as in the picture below:

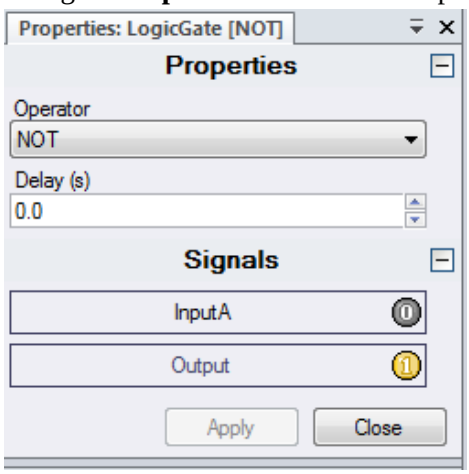


6. Increase the Y value with 10mm. This is to ensure that the sensor is wider than the box and that the entire edge of the box is covered by the sensor so that it is not missed.
7. In the **Axis 1** field type in Z= 100mm (height of the beam).
8. In the **Axis 2** field type in Y= -930mm (width of the beam).
9. Make sure that the sensor is **Active** and that the signal **SensorOut** is low.
10. Press **Apply**.

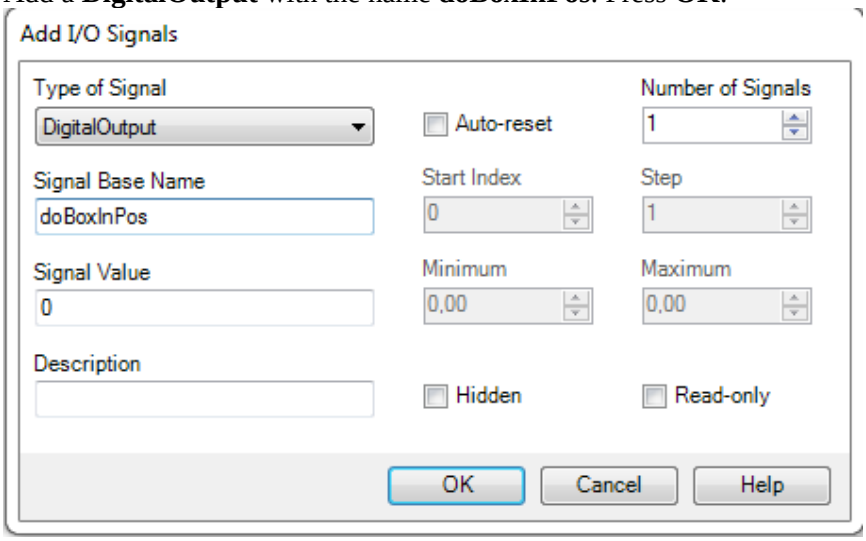


11. In the SC view (SC_inFeeder) go to the **Compose** tab and click the **Add component** button.

12. In the **Signals and Properties** gallery click on the **LogicGate** button.
13. Change the **Operator** to **NOT** in the property window for the **LogicGate**.



14. In the SC view (SC_inFeeder) go to the **Signals and Connections** tab and click the **Add I/O Signals** button.
15. Add a **DigitalOutput** with the name **doBoxInPos**. Press **OK**.



16. Click the **Add I/O Connection** button.
17. Add the I/O connections with the details from the table below.

Source Object	Source Signal	Target Object	Target Signal
PlaneSensor	SensorOut	Queue	Dequeue
PlaneSensor	SensorOut	SC_InFeeder	doBoxInPos
PlaneSensor	SensorOut	LogicGate [NOT]	InputA
LogicGate [NOT]	Output	Source	Execute

Test running and saving the Smart Component

Before we save the SC as a library file we should test run it again.

1. Click on the **my_SC_InFeeder:View1** tab so you can see the conveyor and the box.
2. Open the property window for **SC_InFeeder**
3. Press **Simulation Play**.
4. If a new box not is generated automatically, click on **diNewBox**.
5. Verify that the box stops when it hits the sensor and that **doBoxInPos** goes high.
6. Move the box away from the sensor with **Freehand Move** and verify that **doBoxInPos** goes low and that a new box is generated.
7. Stop the simulation and delete all boxes that were generated during the simulation.
8. Save the **SC_InFeeder** as ... *Libraries \SC_InFeeder.rslib*.

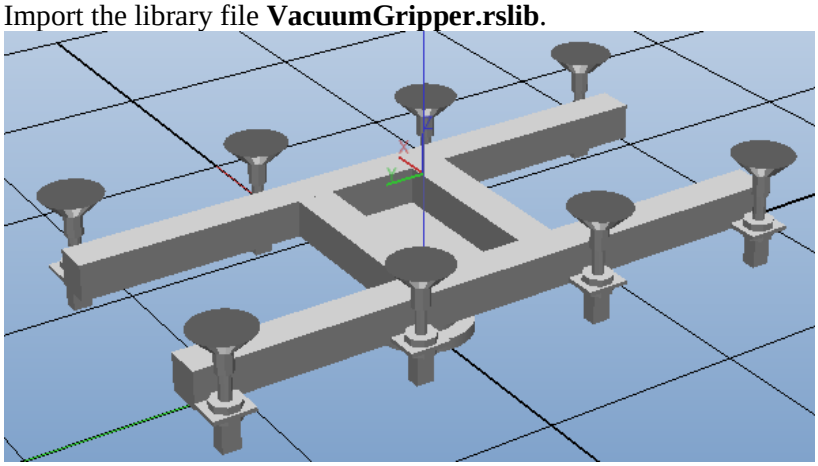
1.2. Create Smart Component - Vacuum Gripper

Overview

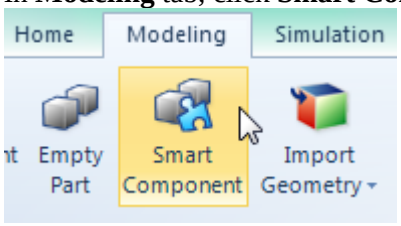
In this exercise we will learn how to create a Smart Component (SC) representing a vacuum gripper.

Preperation

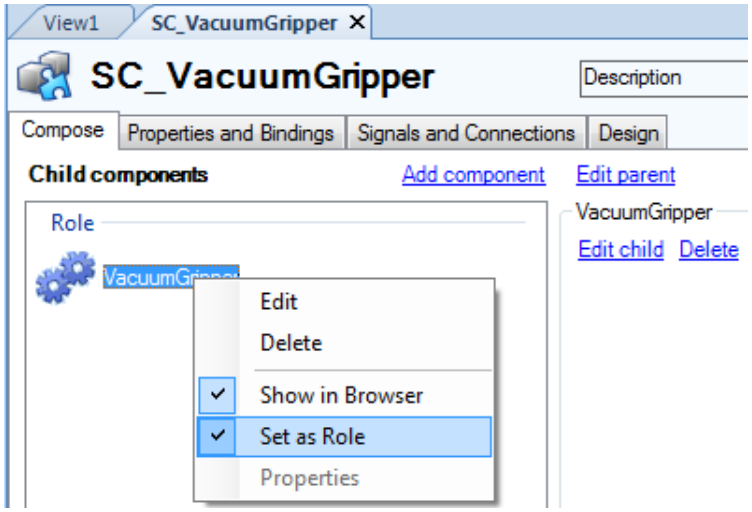
1. Create a new empty station.
2. Import the library file **VacuumGripper.rslib**.



3. As we will modify the library file we need to disconnect the library.
4. In the library context menu, click **Disconnect Library**.
5. In **Modeling** tab, click **Smart Component** to create a new empty SC.



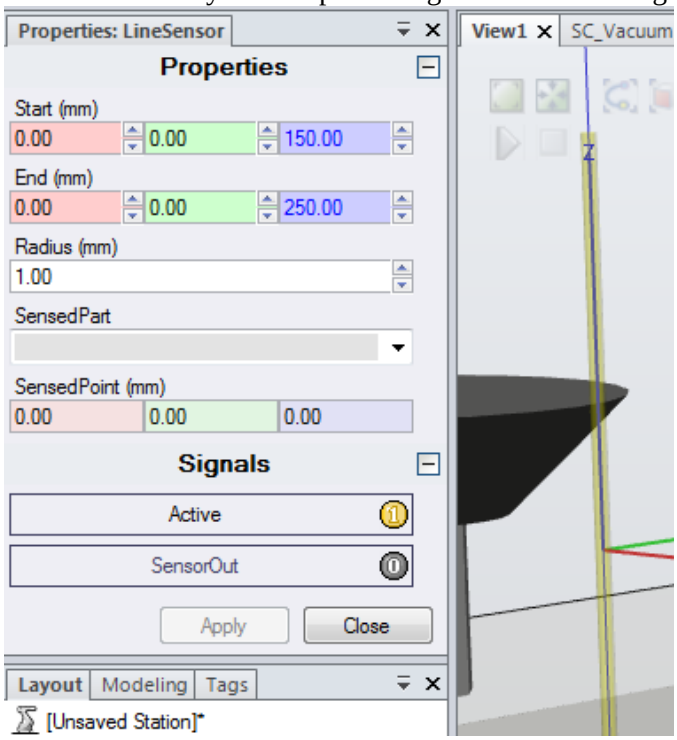
6. Rename the SC to **SC_VacuumGripper**.
7. In Layout browser, drag and drop the vacuum gripper to **SC_VacuumGripper**.
8. In order to get our new component to act as a tool, we need to set the **Vacuum_Gripper** as **role**. In this way the **tooldata** will be created when we attach our **SC gripper** to a robot. In the SC view, set the vacuum gripper as **role** from the conontext menu.



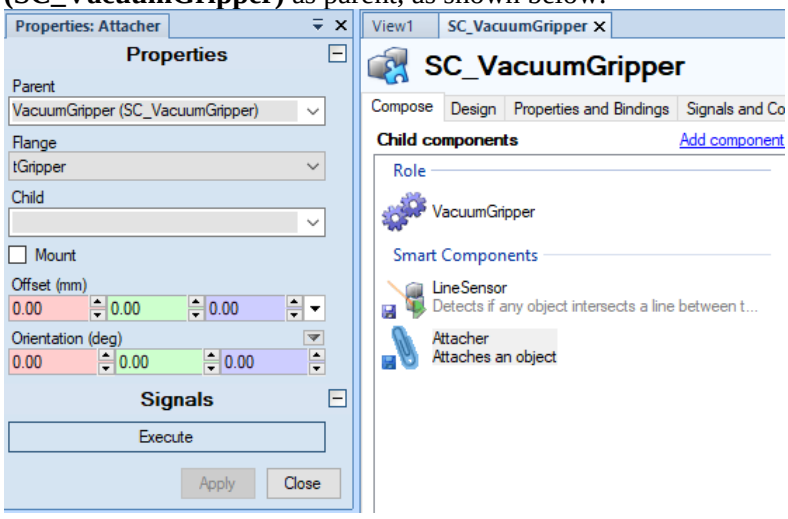
Add Base Components

Now we will start to add base components. We will start with a **sensor** that will react when an object hits the sensor beam.

1. Click **Add component** and then select the **LineSensor** from the **Sensors** gallery.
2. In the **LineSensor** properties window set values as below. After clicking **Apply,** you will see a small cylinder representing the **sensor** in the graphics view.

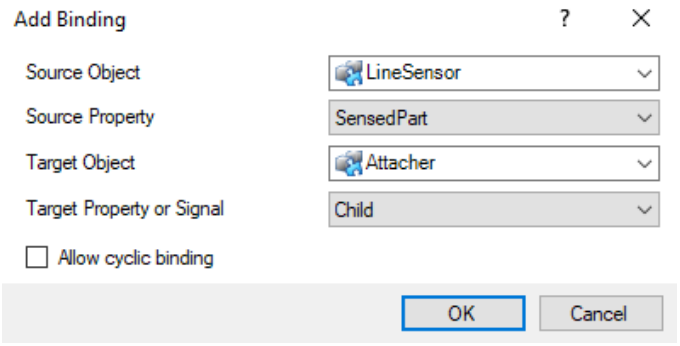


3. Next add an A**ttacher** base component from the **Actions** gallery. When trigged, this base component will attach a **child** component to a **parent** component. The **parent** in this example is represented by the vacuum gripper.
4. In the properties window of the **Attacher**, select the **VacuumGripper (SC_VacuumGripper)** as parent, as shown below.



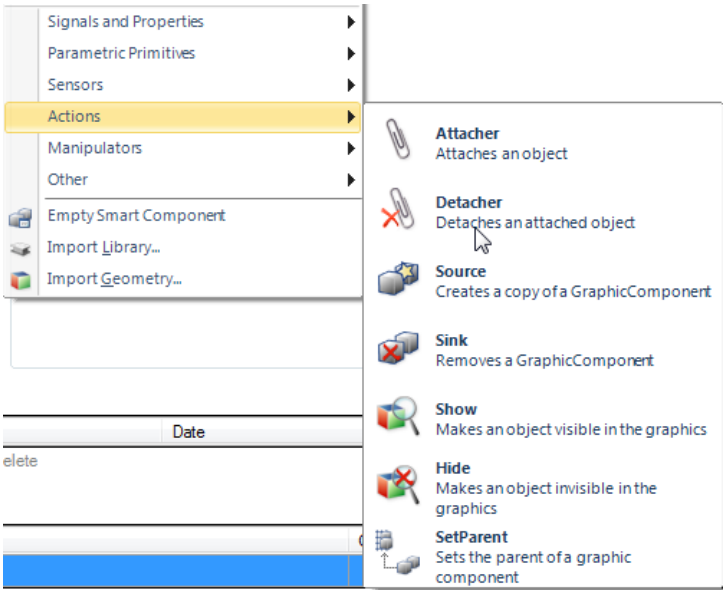
At this point we cannot state which object should represent the child in the **Attacher.** This depends upon the object that is intersected by the sensor in the simulation. Therefore a binding needs to be made between the object sensed by the **LineSensor** we created earlier.

5. Click **Add Binding** in the **Properties and Bindings** tab of the **SC** view. Create the binding as below:

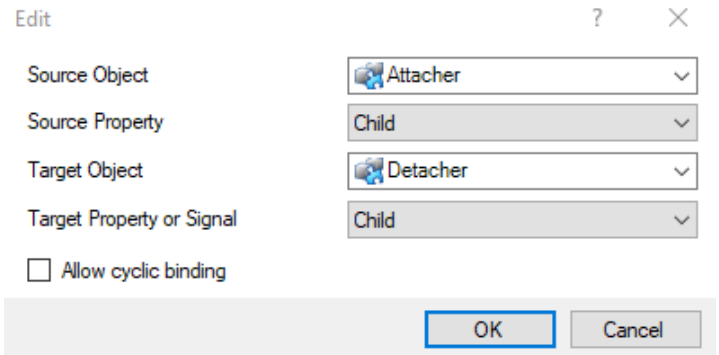


As we also want to be able to detach the attached object we need to add a **Detacher** and a binding between the attached part and the part that will be detached.

6. From the **Compose** tab and add a **Detacher** base component.



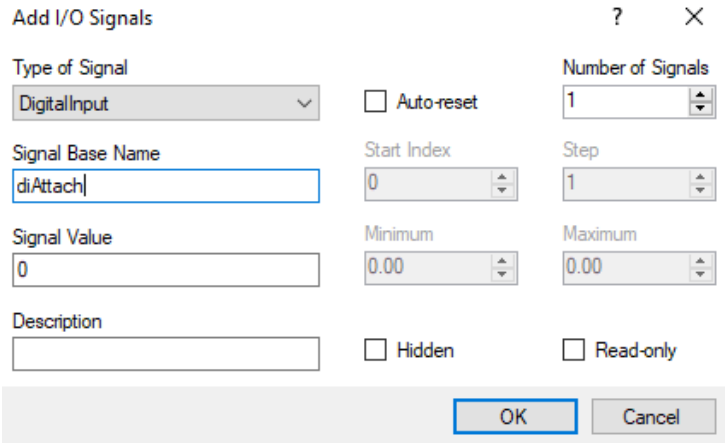
7. Click **Add binding** in the **Properties and Bindings** tab of the SC view again and create the binding as below:



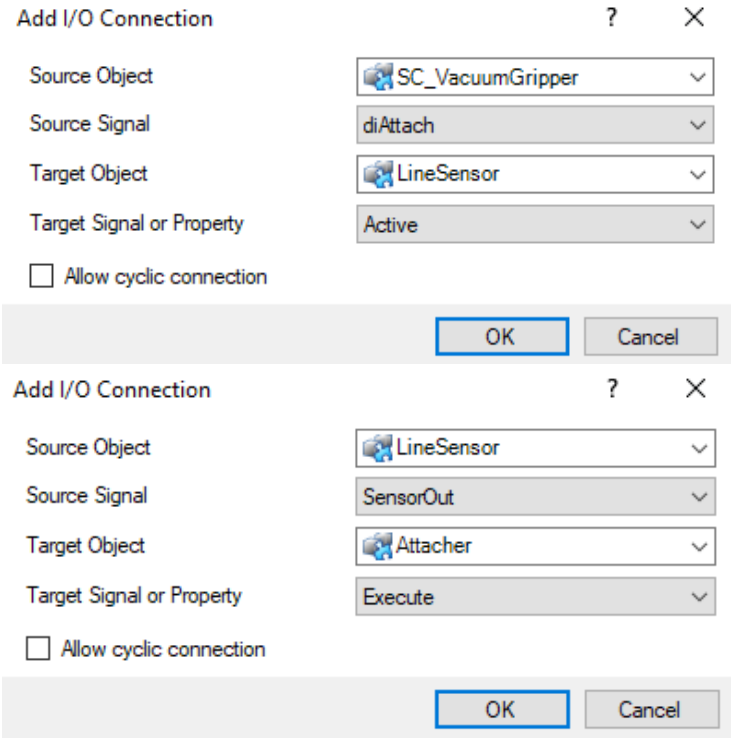
Internal Signals

Next we will define internal signals in our component that later will be cross connected with the **I/O** signals in the Virtual Controller. To define the actions in our **SC**, we also need to define some **I/O** connections.

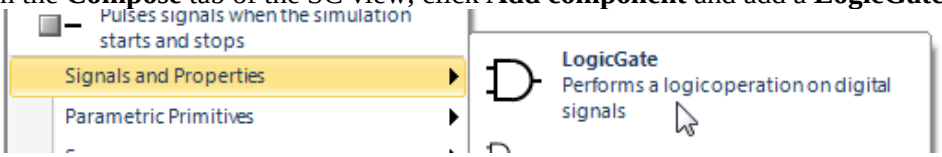
1. In the **Signals and connections** tab of the SC view, click **Add I/O Signals** and add a digital input signal, **diAttach**.



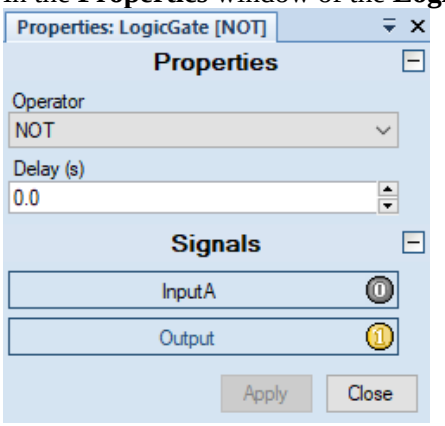
2. Create the first two **I/O connections** with the following input. When the signal is set, the sensor will become active and attach any object breaking the sensor beam.



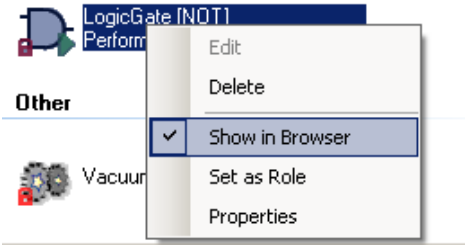
3. When the signal **diAttach** is set low, we want the **Detacher** to execute. Instead of creating a new **I/O** signal we will add a new base component, **LogicGate (NOT).** This will trigger the **Detacher** when the signal is low (**NOT** high).
4. In the **Compose** tab of the SC view, click **Add component** and add a **LogicGate**.



5. In the **Properties** window of the **LogicGate**, change the **Operator** to **NOT**.



6. If you want the component visible in the **Layout** browser, select **Show in Browser** from the context menu.

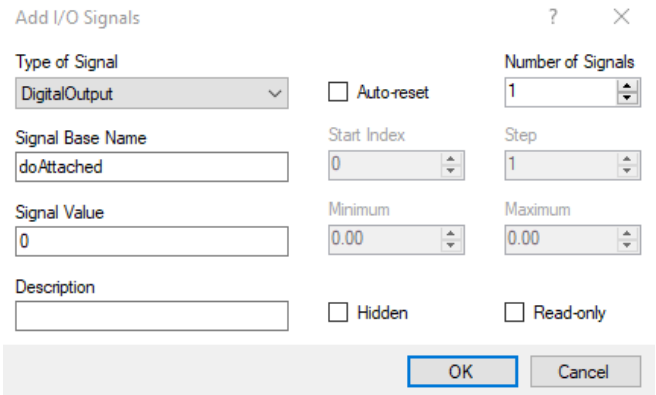


7. Add two more **I/O connections** according to the list below. (Ensure you understand the steps here as this represents the logic discussed in step 3.)

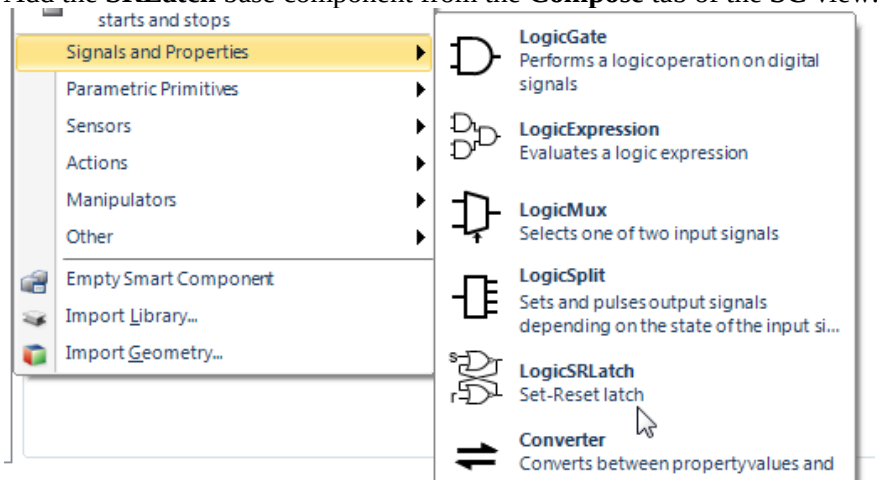
SC_VacuumGripper	diAttach	LogicGate [NOT]	InputA
LogicGate [NOT]	Output	Detacher	Execute

The next step is to create a handshake output signal (SC internal), **doAttached** from our **SC.** This will later be connected to a digital input signal in the robot controller confirming whether something is attached or not. This signal will be controlled by a **SRLatch** (Set-Reset latch) which will be set by the **Attacher** and reset by the **Detacher.**

8. In the **Signals and connections** tab of the **SC** view**,** add a digital output signal, **doAttached.**



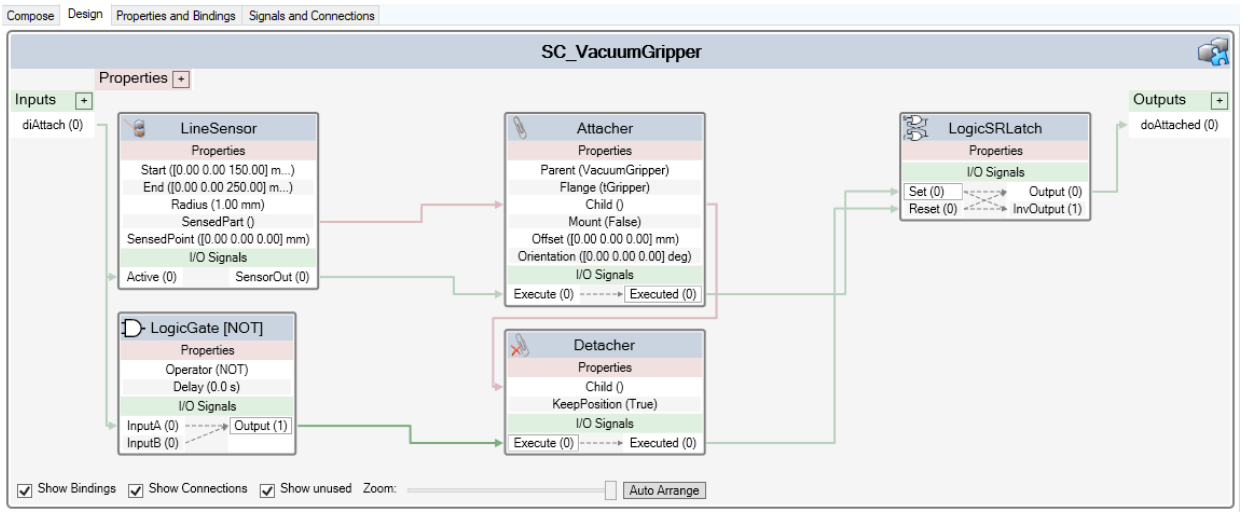
9. Add the **SRLatch** base component from the **Compose** tab of the **SC** view.



10. Then continue adding **I/O connections** according to the list below. Make sure you go through and understand each step. (This represents the logic discussed in step 7.)

Attacher	Executed	LogicSRLatch	Set
Detacher	Executed	LogicSRLatch	Reset
LogicSRLatch	Output	SC_VacuumGripper	doAttached

11. To get an overview of the completed **SC**, click the **Design** tab of the **SC** view. To get a better view click on **Auto Arrange**. Note that you can also drag the various compnents around to arrange them according to your preference. You can also make **I/O connections** and **Bindings** directly in the **Design** tab.



12. Save the **SC_VacuumGripper** as *... Libraries \SC_VacuumGripper.rslib.*

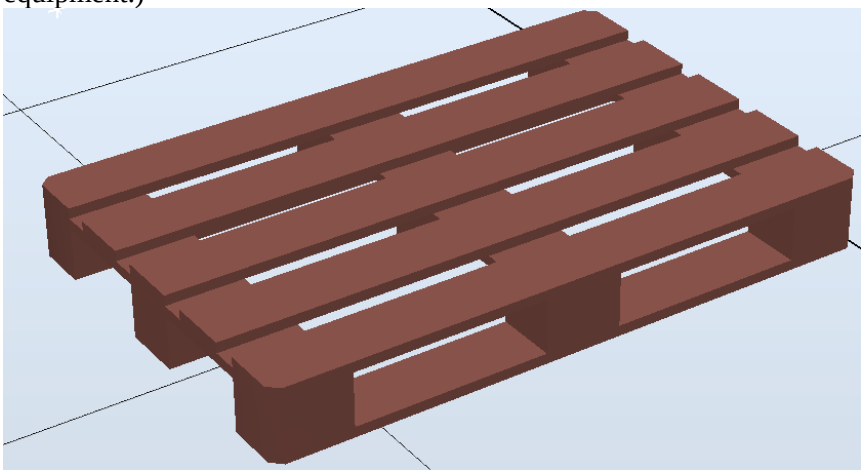
1.3. Create Smart Component - Out Pallet

Overview

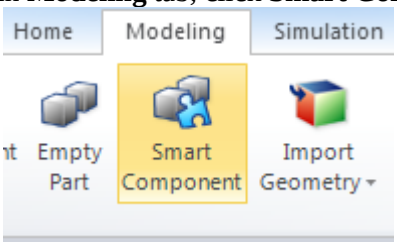
In this exercise we will learn how to create a Smart Component (SC) representing an outfeed pallet.

Preparation

1. Create a new empty station.
2. Import the library file **EuroPallet.rslib**. (Note this is in the standard ABB library under equipment.)



3. Set the pallet's position to X=0, Y=1100, Z=0.
4. As we will modify the library file we need to disconnect the library.
5. In the library context menu, click **Disconnect Library**.
6. In **Modeling** tab, click **Smart Component** to create a new empty SC.

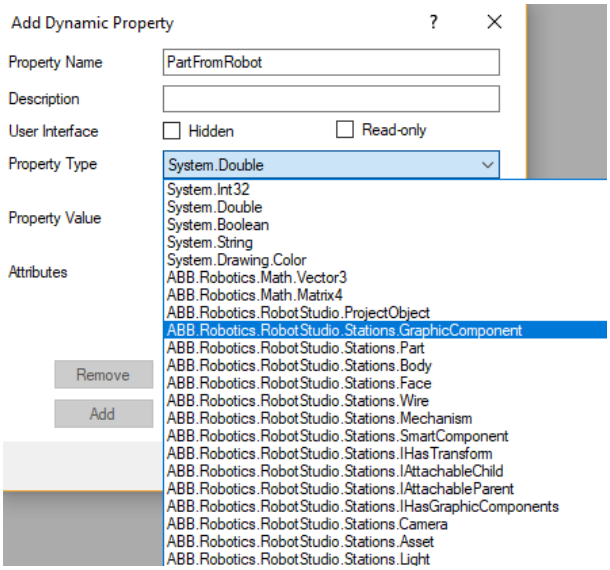


7. Rename the SC to **SC_OutPallet**.
8. In the **Layout** browser, drag and drop the Euro Pallet onto the *SC_OutPallet*.

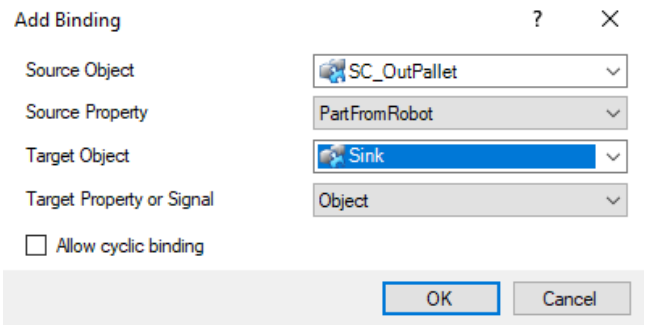
Add Base Components

We will start with a **Sink** for our first base component. This component enable the removal of a graphic component. (In this example it will be the box the robot places on the pallet.)

1. Click **Add component** and then select the **Sink** from the **Actions** gallery.
2. Open up the properties and bindings tab and select add a **Dynamic Property.**
3. Give it the name **PartFromRobot** and under property type select **GraphicComponent.**

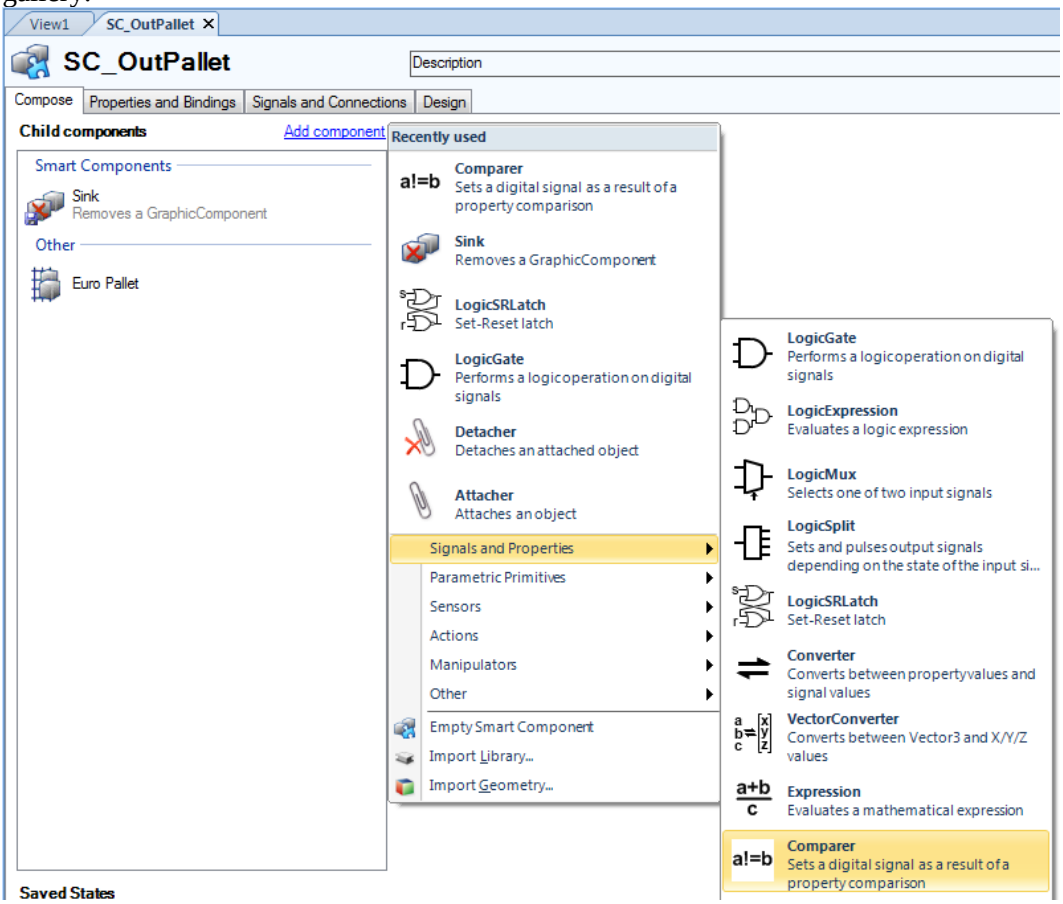


4. Next add a binding to bind the **Sink** with the **Dynamic Property**. (In this case it will be the part dropped off by the robot.)

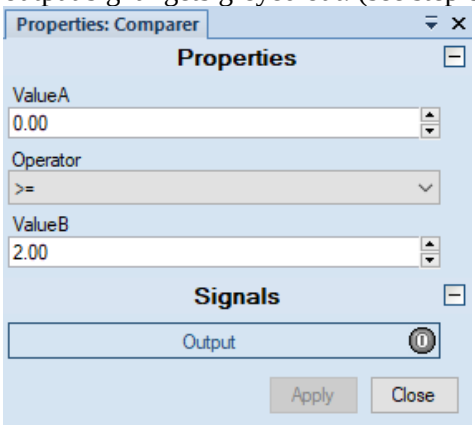


For the next base component we will add a **Comparer**. This component enables the ability to set a signal based upon the result of a property comparison.

5. Add the base component **Comparer** which is found under the **Signal and Properties** gallery.

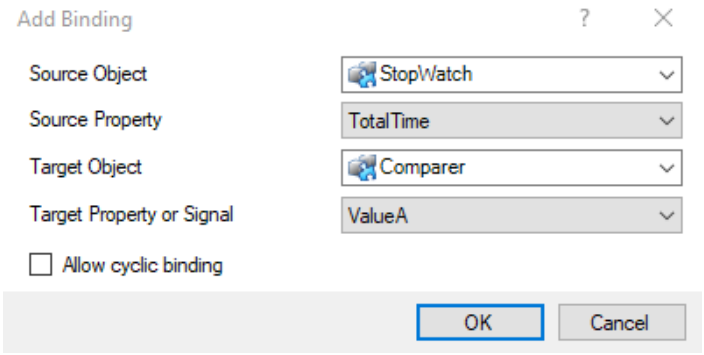


6. In the **Comparer** properties window set values as below and click **Apply**. Note that the output signal gets greyed out. (see step 8)



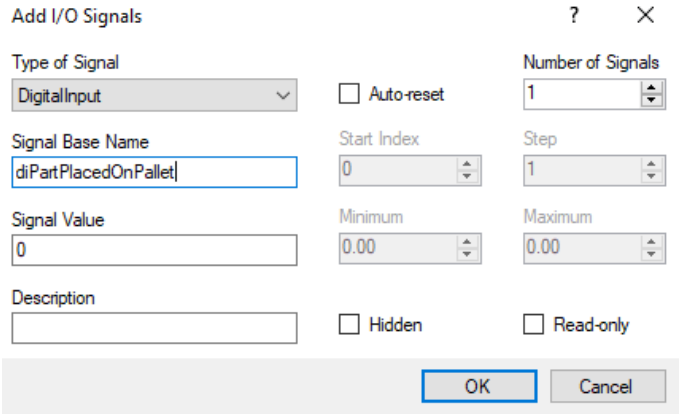
For the next base component we will add a **Stop Watch.**

7. Add a **Stop Watch** which is also found in the **Signals and Properties** gallery.
8. Add a binding to connect the **Stop Watch** to the **Comparer**. In this example by making the binding, the **Comparer** will set an output after 2 seconds of simulation time. Ultimately we will use this to remove/delete the part that will be placed on the pallet.

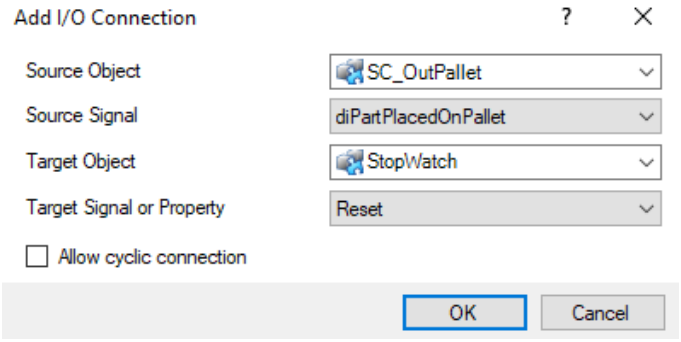


For the next base component we will add a **LogicSRLatch.**

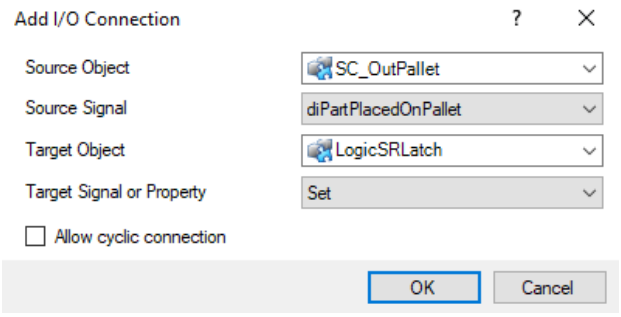
1. Add a **LogicSRLatch** which is again found in the **Signals and Properties** gallery.
2. Next add an I/O signal of the type **DigitalInput** as shown below.



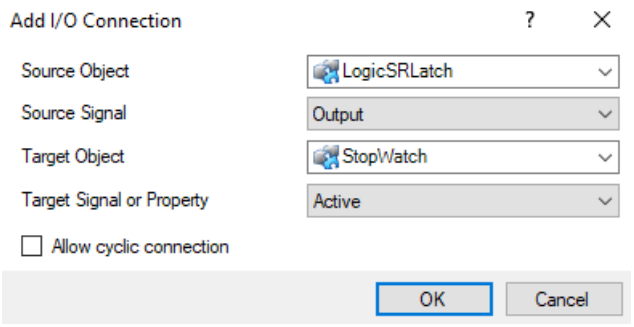
3. Add an **I/O connection** to reset the **Stop Watch** when the digital input goes high.



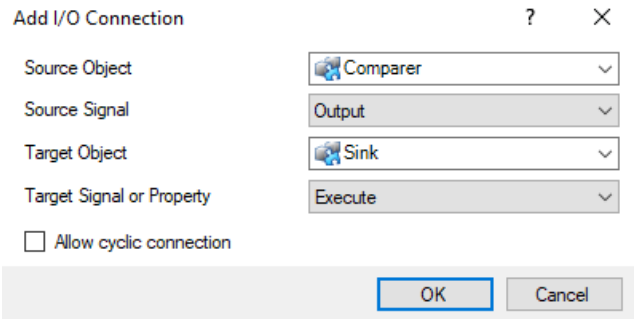
4. Add another **I/O connection** to activate the **LogicSRLatch** when the digital input goes high.



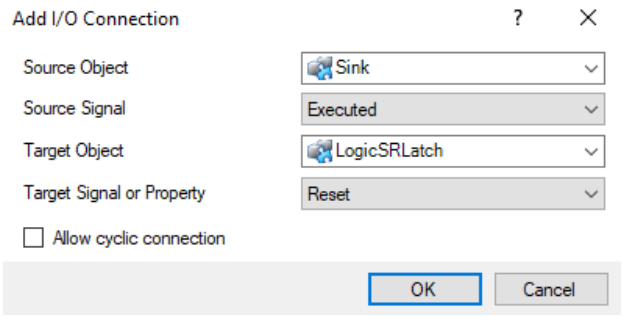
5. Add another **I/O connection** to connect the **LogicSRLatch Output** signal to the **StopWatch Active** signal. This to activate the **StopWatch** when the digital input goes high.



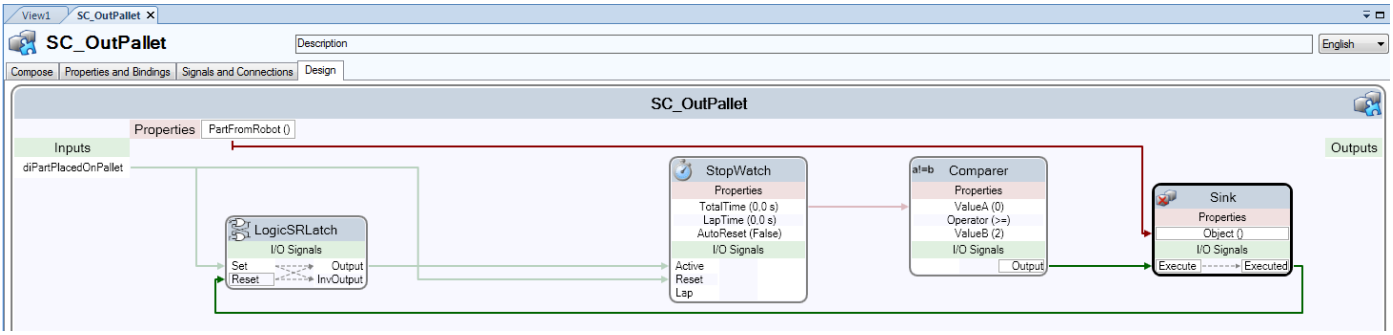
6. Add another **I/O connection** to connect the **Comparer** and the **Sink.** This will delete the object when the **Comparer Output** signal goes high (after 2 seconds).



7. Add the final **I/O connection** to deactivate the **StopWatch** when the object is deleted**.**



8. Verify your design is as shown below.



14. Save the **SC_OutPallet** as ... *Libraries \SC_OutPallet.rslib.*

1.4. Complete the Palletizing Simulation

Overview

In the next exercise we will complete a palletizing simulation by building a station with our smart components along with a robot system.

Preperation

1. Unpack the file *Courseware\Stations\SC_Palletizing_Robot.rspag* to a new folder *Stations\Module_7|my_SC_Palletizing*.
2. Import the ...*Libraries \SC_VacuumGripper.rslib* created earlier in this exercise.
3. Import the ...*Libraries \SC_InFeeder.rslib* created earlier in this exercise.*
4. Import the ...*Libraries \SC_OutPallet.rslib* created earlier in this exercise.
5. Import the library ...*Libraries\box.rslib.**

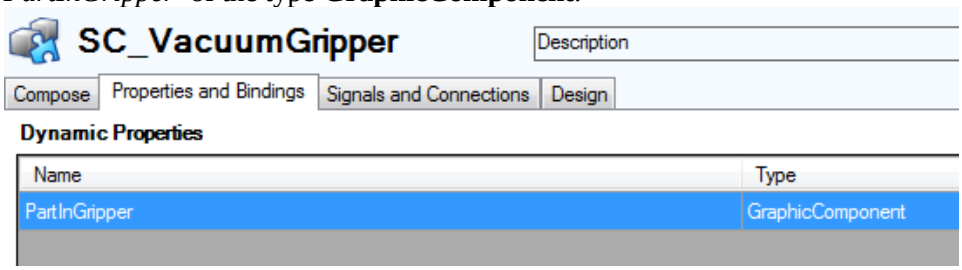
***Note:** When a SC is saved as a library component any links that may have existing during its creation are removed. This gives us the ability to reuse SC in other stations and simulations. When this is done some of the base components may need to be connected, bound etc. to another component.

Hint: In the SC_InFeeder what graphical component are we looking for the **Source** to copy?

Modify the gripper

To be able to delete the object that the robot has placed on the out pallet, we need to modify the Smart Component *SC_VacuumGripper.*

1. Disconnect *SC_VacuumGripper* from library.
2. Right click on *SC_VacuumGripper* and select **Edit Component** in the context menu.
3. In the **Properties and Bindings** tab, add a new **Dynamic Property** with the name *PartInGripper* of the type **GraphicComponent**.

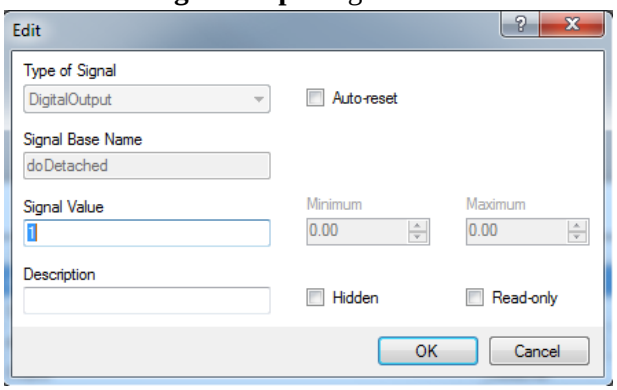


4. Bind the property *LineSensor – SensedPart* to the new property.

Property Bindings

Source Object	Source Property	Target Object	Target Property
LineSensor	SensedPart	Attacher	Child
Attacher	Child	Detacher	Child
LineSensor	SensedPart	SC_VacuumGripper	PartInGripper

5. Add a new **DigitalOutput** signal with an initial value of 1 and name it *doDetached*.



6. Make an **I/O Connection** from the *LogicSRLatch –InvOutput* to *doDetached.*

I/O Connections

Source Object	Source Signal	Target Object	Target Signal
SC_VacuumGripper	diAttach	LineSensor	Active
LineSensor	SensorOut	Attacher	Execute
SC_VacuumGripper	diAttach	LogicGate [NOT]	InputA
LogicGate [NOT]	Output	Detacher	Execute
Attacher	Executed	LogicSRLatch	Set
Detacher	Executed	LogicSRLatch	Reset
LogicSRLatch	Output	SC_VacuumGripper	doAttached
LogicSRLatch	InvOutput	SC_VacuumGripper	doDetached

7. Close the Smart Component Editor window and save the gripper with the same name as before.
8. Attach *SC_VacuumGripper* to the robot.

Station Logic setup

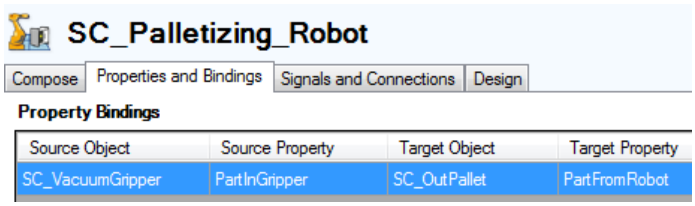
The last step is to setup I/O connections between the Smart Components and the Virtual Controller to get a complete simulation. This can be thought of as "virtual wiring" between the various components in the simulation environment whereas in the real application there would be physical wiring in place between the actual components.

1. Setup the I/O connections according to the list below.

I/O Connections

Source Object	Source Signal	Target Object	Target Signal
SC_VacuumGripper	doDetached	SC_OutPallet	diPartPlacedOnPallet
IRB6620_SC_system	doNewBox	SC_InFeeder	diNewBox
SC_InFeeder	doBoxInPos	IRB6620_SC_system	diBoxInPos
IRB6620_SC_system	doVacuum	SC_VacuumGripper	diAttach
SC_VacuumGripper	doAttached	IRB6620_SC_system	diVacuum

2. Add a binding from *SC_VacuumGripper - partInGripper* to *SC_OutPallet - PartFromRobot*.



Run the Simulation

1. Make sure all I/O signals (including all SC) have the correct start values***** and save the current state.
2. Add the saved state in **Simulation Setup** and then run the simulation. Refer to Module 5 for assistance if required.
3. Save the station as a *station file* (.rsstn) and as a *Pack and Go* file (.rspag)

***Note:** Depending upon the exact signal values you start with multiple boxes may be created when the simulation is started. One option to correct this may be to change the Rapid code. In the main procedure an I/O is being pulsed in order to create a new box. The SC_InFeeder is also using a logic gate to create a new part when the sensor is low at the end of the conveyor.

```
21      PROC main()
22          SetDO doVacuum,0;
23          WaitDI diVacuum,0;
24      !    PulseDO doNewBox;
25          WHILE TRUE DO
26              PickLeave;
27          ENDWHILE
28      ENDPROC
```

© Copyright 2017 ABB All right reserved.
ABB AB
Robotics Products
SE-721 68 Västerås
Sweden

6.3 Utvalg av produkter

Plukkfordeling dagens artikler

- Volumfordeling hentet fra periode 08.08.2019 – 28.01.2020, uten skoleferie.

Artikler	Beskrivelse	%
358	Kakao	29,7 %
172	Lettmelk 1% fett	23,8 %
213	Lettmelk 0,5% fett	19,8 %
5688	Eple fruktdrikk	8,5 %
5972	Smoothie	6,8 %
6601	Lettmelk med jordbærsmak	2,6 %
6599	Appelsindrikk	2,3 %
4529	Laktosefri lettmelk	1,6 %
6128	Epler i biter	1,1 %
6129	Gulrøtter	1,1 %
5956	Yoghurt Vanilje	0,9 %
5960	Havregrøt med jordbærsaus	0,6 %
6600	Øko lettmelk	0,6 %
6602	Havredrikk	0,3 %
6603	Jordbæryogurt med topping	0,1 %

Eksempler lastbærer og D-pak

Artikkel 213 (RC og D-pak)

Eksempler lastbærer og D-pak

Artikkel 5956 (Pall og D-pak)

Eksempler lastbærer og D-pak

Artikkel 5960 (Pall og D-pak)

Eksempler lastbærer og D-pak

Artikkel 6602 (Pall og D-pak)

Eksempler lastbærer og D-pak

Artikkel 4529 (Pall og D-pak)

Eksempel D-pak

Artikkel 6128

Eksempel D-pak

Artikkel 6129

6.4 Bildebehandlingskode

```
import cv2
import numpy as np

# Ender img fra rek12 og rek13
img = cv2.imread('rek12.jpg')

scale_percent = 20
width = int(img.shape[1] * scale_percent / 100)
height = int(img.shape[0] * scale_percent / 100)
dim = (width, height)
resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)

hsv_frame = cv2.cvtColor(resized, cv2.COLOR_BGR2HSV)
low = np.array([0,0,183])
high = np.array([137,35,249])

mask = cv2.inRange(hsv_frame, low, high)

contours, hierarchy = cv2.findContours(mask, cv2.RETR_TREE,
                                       cv2.CHAIN_APPROX_SIMPLE)

for cnt in contours:
    x, y, w, h = cv2.boundingRect(cnt)
    if w > 50 and h > 30:
        if w < 800 and h < 800:
            size = cv2.contourArea(cnt)
            if size > 10000:
                if x < 111:
                    if size > 12000:
                        rek = cv2.rectangle(resized,(x,y),(x+w,y+h)
    ,(0,215,255),2)
                        print('Esken ligger p   langsiden: Lag-A')
                        Lag = 1
                        print(size)
                    else:
                        rek = cv2.rectangle(resized,(x,y),(x+w,y+h)
    ,(0,215,255),2)
                        print('Esken ligger p   kortsiden: Lag-B')
                        print(size)
                        Lag = 0


cv2.imshow('img', resized)
#cv2.imshow('test', mask)
cv2.waitKey(0)
cv2.destroyAllWindows()

# trykker q for    komme videre


```

```
if Lag == 0:
    image1 = cv2.imread('Sirkel_kort.jpg')
    image = cv2.flip(image1, 1)

    output = image.copy()

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray_blurred = cv2.blur(gray, (3, 3))

    circles = cv2.HoughCircles(gray_blurred, cv2.HOUGH_GRADIENT, 1, 20,
    param1 = 50,
                           param2 = 30, minRadius = 23, maxRadius = 25)


    if circles is not None:
        max_x = 150
        min_x = 0
        max_y = 450
        min_y = 50
        circles = np.round(circles[0, :]).astype("int")
        for (x, y, r) in circles:
            if min_x < x < max_x and min_y < y < max_y:
                b = cv2.circle(output, (x, y), r, (0, 255, 255), 2)
        for i in circles:
            if min_y < i[1] < max_y and min_x < i[0] < max_x:
                b = cv2.circle(output,(i[0],i[1]),2,(0,0,255),3)
                print(i)
        cv2.imshow("circle",output)
        cv2.waitKey(0)
        # NESTE ESKE
        output = image.copy()
        for (x, y, r) in circles:
            if (min_x + 150) < x < (max_x + 150) and min_y < y < max_y:
                b = cv2.circle(output, (x, y), r, (0, 255, 255), 2)
        for i in circles:
            if min_y < i[1] < max_y and (min_x + 150) < i[0] < (max_x +
    150):
                b = cv2.circle(output,(i[0],i[1]),2,(0,0,255),3)
                print(i)
        cv2.imshow("circle",output)
        cv2.waitKey(0)

else:
    image1 = cv2.imread('sirkler_riktig.jpg')
    image = cv2.flip(image1, 1)

    output = image.copy()

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray_blurred = cv2.blur(gray, (3, 3))

    circles = cv2.HoughCircles(gray_blurred, cv2.HOUGH_GRADIENT, 1, 20,
```

```
    param1 = 50,
                            param2 = 30, minRadius = 23, maxRadius = 25)

    if circles is not None:
        max_x = 550
        min_x = 100
        max_y = 450
        min_y = 300
        circles = np.round(circles[0, :]).astype("int")
        for (x, y, r) in circles:
            if min_x < x < max_x and min_y < y < max_y:
                a = cv2.circle(output, (x, y), r, (0, 255, 255), 2)
        for i in circles:
            if min_y < i[1] < max_y and min_x < i[0] < max_x:
                b = cv2.circle(output,(i[0],i[1]),2,(0,0,255),3)
                print(i)
        cv2.imshow("circle",output)
        cv2.waitKey(0)
        #NESTE ESKE
        output = image.copy()
        for (x, y, r) in circles:
            if min_x < x < (max_x + 100) and (min_y - 150) < y < (max_y
    -150):
                a = cv2.circle(output, (x, y), r, (0, 255, 255), 2)
        for i in circles:
            if (min_y - 150) < i[1] < (max_y-150) and min_x < i[0] < (
    max_x + 100):
                b = cv2.circle(output,(i[0],i[1]),2,(0,0,255),3)
                print(i)
        cv2.imshow("circle",output)
        cv2.waitKey(0)
```

Listing 6.1: Bildebehandling

6.5 Bildebehandlingskode - HSV

```
import cv2
import numpy as np

img = cv2.imread('rek10.jpg')

scale_percent = 20
width = int(img.shape[1] * scale_percent / 100)
height = int(img.shape[0] * scale_percent / 100)
dim = (width, height)
resized = cv2.resize(img, dim, interpolation = cv2.INTER_AREA)

def nothing(x):
    # any operation
    pass

cv2.namedWindow("Trackbars")
```

```
cv2.createTrackbar("L-H", "Trackbars", 0, 180, nothing)
cv2.createTrackbar("L-S", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("L-V", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("U-H", "Trackbars", 180, 180, nothing)
cv2.createTrackbar("U-S", "Trackbars", 255, 255, nothing)
cv2.createTrackbar("U-V", "Trackbars", 255, 255, nothing)

while True:
    hsv_frame = cv2.cvtColor(resized, cv2.COLOR_BGR2HSV)

    l_h = cv2.getTrackbarPos("L-H", "Trackbars")
    l_s = cv2.getTrackbarPos("L-S", "Trackbars")
    l_v = cv2.getTrackbarPos("L-V", "Trackbars")
    u_h = cv2.getTrackbarPos("U-H", "Trackbars")
    u_s = cv2.getTrackbarPos("U-S", "Trackbars")
    u_v = cv2.getTrackbarPos("U-V", "Trackbars")


    low = np.array([l_h,l_s,l_v])
    high = np.array([u_h,u_s,u_v])

    mask = cv2.inRange(hsv_frame, low, high)


    cv2.imshow("Frame", resized)
    cv2.imshow("HSV", mask)


    if cv2.waitKey(1) & 0xFF == ord('q'):
        break
```

Listing 6.2: Bildebehandling - HSV

6.6 Bilder

Figur 6.1: Bilde av Lag-A, sett ovenifra

Figur 6.2: Bilde av Lag-A, sett fra siden

Figur 6.3: Bilde av Lag-B, sett ovenifra

Figur 6.4: Bilde av Lag-B, sett fra siden

6.7 Kode til simulering

```
MODULE Module1
        CONST robtarget Target_10
   :=[[264.034808248,28.54404501,958.376172361]
        ,[0.010947783,-0.837143251,-0.006762853,
        0.54683232],[0,-1,0,1],[1839.327760425,9E+09,9E+09,9E+09,9E
   +09,9E+09]];
    CONST robtarget Target_50
   :=[[-118.566865171,454.763117793,399.604515449],
    [0.005656405,0.999900009,0.012932419,
    -0.000854407],[0,0,-1,0],[1143.405062362,9E+09,9E+09,9E+09,9E+09,9E
   +09]];
    CONST robtarget Target_60
   :=[[-118.566886767,454.763125542,309.976556692],
    [0.005656407,0.999900009,0.012932413,
    -0.000854401],[0,0,-1,0],[1143.405062362,9E+09,9E+09,9E+09,9E+09,9E
   +09]];
    CONST robtarget Target_70
   :=[[-252.42070582,1104.624979027,608.133891366],
    [0.705510059,0.708581382,0.008616692,
    -0.009681668],[0,0,-1,0],[1009.55127455,9E+09,9E+09,9E+09,9E+09,9E
   +09]];
    CONST robtarget Target_80
   :=[[-258.248688455,708.574556598,404.359025546],
    [0.499410885,-0.500429715,0.49507453,
    0.50503473],[0,-2,1,0],[1009.55127455,9E+09,9E+09,9E+09,9E+09,9E
   +09]];
    CONST robtarget Target_90
   :=[[-1267.799940327,1216.529042861,730.092523041],
    [0.49941097,-0.500429745,0.495074501,
    0.505034644],[0,-2,1,0],[0,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_100
   :=[[-952.04910864,-36.375607424,1104.040021551],
    [0.14087316,0.6899836,0.139467422,
    -0.696151006],[0,-3,2,0],[0,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_110
   :=[[-952.04910864,-36.375607424,1104.040021551],
    [0.14087316,0.6899836,0.139467422,
    -0.696151006],[0,-3,2,0],[0,9E+09,9E+09,9E+09,9E+09,9E+09]];
    CONST robtarget Target_120
   :=[[2942.777152257,-36.375607424,1104.040021551],
    [0.14087316,0.6899836,0.139467422,
    -0.696151006],[0,-3,2,0],[3894.826260898,9E+09,9E+09,9E+09,9E+09,9E
   +09]];
    CONST robtarget Target_190
   :=[[1744.154458009,-32.302485396,533.989833653],
    [0.181081979,0.869887016,0.085177374,
    -0.450833351],[0,-4,3,0],[4025.423532037,9E+09,9E+09,9E+09,9E+09,9E
   +09]];
    CONST robtarget Target_200
   :=[[1744.154425237,479.239798289,526.336987965],
```

```
33      [0.181082104,0.869887007,0.085177202,
34      -0.450833351],[0,-4,3,0],[4025.423532037,9E+09,9E+09,9E+09,9E+09,9E
        +09]];
35      CONST robtarget Target_210
        :=[[1744.154447356,479.239486451,526.336927767],
36      [0.005204977,-0.999736303,-0.001432693,
37      0.022319928],[0,-5,3,0],[4025.423532037,9E+09,9E+09,9E+09,9E+09,9E
        +09]];
38      CONST robtarget Target_220
        :=[[1744.154416166,932.19185816,909.892520453],
39      [0.005204867,-0.999736304,-0.001432651,
40      0.022319946],[0,-4,3,0],[4025.423532037,9E+09,9E+09,9E+09,9E+09,9E
        +09]];
41      CONST robtarget Target_250
        :=[[1709.586248803,743.073593354,909.892529174],
42      [0.016180056,-0.011480056,-0.702181538,
43      -0.711721506],[0,-2,2,0],[1890.16475106,9E+09,9E+09,9E+09,9E+09,9E
        +09]];
44      CONST robtarget Target_380
        :=[[1574.687631058,743.073910406,666.947109627],
45      [0.492008284,-0.502123556,0.501246954,
46      0.504530746],[0,-2,1,0],[1890.164852142,9E+09,9E+09,9E+09,9E+09,9E
        +09]];
47      CONST robtarget Target_390
        :=[[-685.376871799,249.439148416,1346.437851298],
48      [0.077467558,0.706054362,0.069744307,
49      -0.700443965],[0,-3,2,0],[0,9E+09,9E+09,9E+09,9E+09,9E+09]];
50      CONST robtarget Target_400
        :=[[-1463.541039439,1270.563963782,1054.602920023],
51      [0.077467609,0.706054524,0.069744191,
52      -0.700443807],[0,-3,2,0],[0,9E+09,9E+09,9E+09,9E+09,9E+09]];
53      CONST robtarget Target_410
        :=[[-1029.414093515,133.371341227,1145.889957874],
54      [0.154185122,0.694068372,0.147127584,
55      -0.687640544],[0,-3,2,0],[0,9E+09,9E+09,9E+09,9E+09,9E+09]];
56      CONST robtarget Target_420
        :=[[2975.283734039,133.371327258,1497.18698523],
57      [0.154185046,0.694068382,0.147127719,
58      -0.687640521],[0,-3,2,0],[4004.697805367,9E+09,9E+09,9E+09,9E+09,9E
        +09]];
59      CONST robtarget Target_430
        :=[[2975.283699817,294.05659913,951.871854862],
60      [0.010745912,0.999932736,-0.004360369,
61      0.000187782],[0,-4,3,0],[4004.697799683,9E+09,9E+09,9E+09,9E+09,9E
        +09]];
62      CONST robtarget Target_550
        :=[[2975.283720175,955.297599517,1348.240958349],
63      [0.010745918,0.999932736,-0.004360329,
64      0.00018777],[0,-4,3,0],[4004.697799683,9E+09,9E+09,9E+09,9E+09,9E
        +09]];
65      CONST robtarget Target_560
        :=[[3000.914670194,818.653328608,1348.24090451],
```

```
66      [0.007175437,-0.002743226,-0.719177371,
67      -0.694784065],[0,-2,2,0],[3185.518485301,9E+09,9E+09,9E+09,9E+09,9E
        +09]];
68      CONST robtarget Target_450
        :=[[2877.859851603,1054.924997438,1087.825955093],
69      [0.005135452,0.707822819,0.013259121,
70      -0.706246897],[0,-3,2,0],[4004.697799683,9E+09,9E+09,9E+09,9E+09,9E
        +09]];
71      CONST robtarget Target_580
        :=[[2877.859796074,356.066966535,1340.398985048],
72      [0.005135542,0.707822863,0.013259177,
73      -0.706246852],[0,-3,2,0],[4004.697799683,9E+09,9E+09,9E+09,9E+09,9E
        +09]];
74      CONST robtarget Target_460
        :=[[-1126.838010308,-42.711774275,1636.051003352],
75      [0.005135142,0.707822831,0.013259474,
76      -0.70624688],[1,-2,1,0],[0,9E+09,9E+09,9E+09,9E+09,9E+09]];
77      CONST robtarget Target_470
        :=[[-1510.836035103,1317.38397017,755.813412352],
78      [0.00513524,0.707822895,0.013259428,
79      -0.706246816],[0,-3,2,0],[0,9E+09,9E+09,9E+09,9E+09,9E+09]];
80      CONST robtarget Target_480
        :=[[-1165.553010258,82.66339674,1113.518933336],
81      [0.160715499,0.687482753,0.168987038,
82      -0.687736413],[0,-3,2,0],[0,9E+09,9E+09,9E+09,9E+09,9E+09]];
83      CONST robtarget Target_490
        :=[[3863.364248297,82.663405913,1607.730955733],
84      [0.011083183,0.999874834,0.005646565,
85      0.009777262],[0,-4,3,0],[5028.917267654,9E+09,9E+09,9E+09,9E+09,9E
        +09]];
86      CONST robtarget Target_500
        :=[[3863.364252904,82.663400868,484.433563973],
87      [0.011083198,0.999874834,0.00564657,
88      0.009777237],[0,-4,3,0],[5028.917267654,9E+09,9E+09,9E+09,9E+09,9E
        +09]];
89  !*********************************************************
90      !
91      ! Module:  Module1
92      !
93      ! Description:
94      !   <Insert description here>
95      !
96      ! Author: sande
97      !
98      ! Version: 1.0
99      !
100     !*********************************************************
101
102
103     !*********************************************************
104     !
105     ! Procedure main
```

```
    !
    !   This is the entry point of your program
    !
    !*********************************************************
    PROC main()
        Path_20;
    ENDPROC
    PROC Path_20()
        MoveL Target_110,v1000,z100,tool0\WObj:=wobj0;
        MoveL Target_120,v1000,z100,tool0\WObj:=wobj0;
        MoveL Target_190,v1000,z100,tool0\WObj:=wobj0;
        MoveL Target_200,v500,z100,tool0\WObj:=wobj0;
        MoveL Target_210,v500,z100,tool0\WObj:=wobj0;
        WaitTime 5;
        MoveL Target_220,v100,z100,tool0\WObj:=wobj0;
        MoveL Target_250,v1000,z100,tool0\WObj:=wobj0;
        WaitTime 5;
        MoveL Target_380,v500,z100,tool0\WObj:=wobj0;
        WaitTime 5;
        MoveL Target_390,v500,z100,tool0\WObj:=wobj0;
        MoveL Target_400,v500,z100,tool0\WObj:=wobj0;
        WaitTime 5;
        MoveL Target_410,v500,z100,tool0\WObj:=wobj0;
        WaitTime 5;
        MoveL Target_420,v500,z100,tool0\WObj:=wobj0;
        MoveL Target_430,v500,z100,tool0\WObj:=wobj0;
        WaitTime 5;
        MoveL Target_550,v1000,z100,tool0\WObj:=wobj0;
        MoveL Target_560,v1000,z100,tool0\WObj:=wobj0;
        WaitTime 5;
        MoveL Target_450,v100,z100,tool0\WObj:=wobj0;
        WaitTime 2;
        MoveL Target_580,v1000,z100,tool0\WObj:=wobj0;
        MoveL Target_460,v1000,z100,tool0\WObj:=wobj0;
        MoveL Target_470,v1000,z100,tool0\WObj:=wobj0;
        WaitTime 3;
        MoveL Target_480,v100,z100,tool0\WObj:=wobj0;
        WaitTime 3;
        MoveL Target_490,v1000,z100,tool0\WObj:=wobj0;
        MoveL Target_500,v500,z100,tool0\WObj:=wobj0;
    ENDPROC
ENDMODULE
```

Listing 6.3: Kode i Rapid for simulering av roboten

6.8 Kode til kommunikasjon

```
robot = RWS.RWS("")

robot.set_rapid_variable("action", action)
robot.set_rapid_variable("image_processed", image_processed)
wait_for_rapid(robot, "ready_flag")
```

```
6
7  def wait_for_rapid (self, var='ready_flag'):
8
9
10     while self.get_rapid_variable(var) == "FALSE" and self.is_running()
       :
11
12         time.sleep(0.1)
13         self.set_rapid_variable(var, "FALSE")
```

Listing 6.4: Kommunikasjon mellom Python og RobotStudio

6.9 Kode til sugekopper

```
1  PROC main()
2      SetDO doVacuum, 0;
3      WaitDI diVacuum, 0;
4
5      WHILE TRUE DO
6          PickPack;
7      ENDWHILE
8  ENDPROC
```

Listing 6.5: Kode i Rapid for simulering av roboten