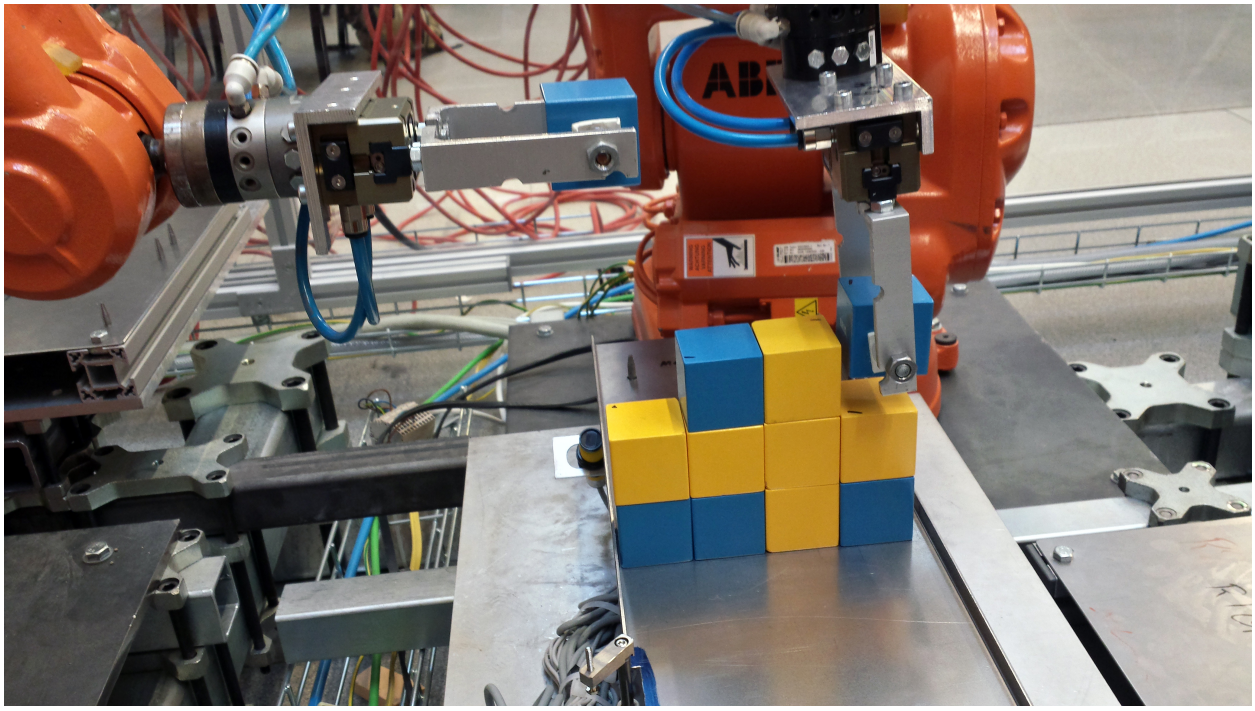
CHALMERS



Den flexibla klossfabriken

En utvärdering av operationskonceptet för produktion av modulära produkter.

Kandidatarbete inom Signaler och System, SSYX02-16-81

Marcus Andersson
Kristian Eide
Dandan Ge
Tobias Glans
Jacob Henricsson
Atle Zvantesson

Institutionen för Signaler och System
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2016

Förord

Den här rapporten är resultatet av ett kandidatarbet som genomförts på Chalmers Tekniska Högskola under vårterminen 2016. Vi skulle vilja tacka följande personer:

- Vår handledare *Kristofer Bengtsson* för bra ledning och stöd
- *Martin Dahl* för hjälp med *Sequence Planner* och optimering
- Vår examinator *Petter Falkman*
- *Per Nyqvist* för hjälp med robotarna
- *Hans Sjöberg* för hjälp till att iordningsställa cellen
- *Jan Bragee* och *Reine Nohlborg* från Chalmers experimentverkstad för att ni hjälpte oss med att tillverka byggplattor och paletter
- Vår samarbetsgrupp *SSYX02-16-01* för ett gott samarbete

Abstract

Today, the demand for custimized products are growing at a steady rate. This puts a lot of stress on production industries, as factory-lines needs to be changed on order to meet the market demands. This project will examine if it is possible to use a operation-based concept in order to make manufacturing more flexible, in order to avoid costly restarts, manufacturer restructures and implementation of new code.

The project has been divided into three parts in order to make it more manageble: A high-level control system that takes care of all the needed plans and calculations; A PLC controlling all signals coming in and out of the working area; And finally a part that handles the robots and automation that operates in the cell.

In order to practically test automated production of a customizable product, a small production cell simulating an industrial production line is used to build towers of colored building blocks.

The building process starts with an operator defining the layout of a custom tower. Instruction detaling how the blocks should be placed is then given to the operator. The blocks are then brought into the cell where robots builds the tower according to instructions supplied by the high-level control system.

The project has been performed in collaboration with the group *SSYX02-16-01*, who have written a complementary report.

Sammanfattning

I dagens samhälle är det allt fler som vill ha specialtillverkade produkter, vilket är tydligt i till exempel bilindustrin. Detta lägger stor press på industrin eftersom fabrikslinor ofta måste ändras för att kunna klara av de nya tillverkningsprocesserna som kan krävas. Det här projektet går ut på att utvärdera om det går att använda operationskoncept för att få tillverkningen mer flexibel och minska tiden som läggs på återstart, omställning och implementering av ny kod.

Projektet delas in i tre delar: En övergripande styrning som sköter den planering och beräkning som behövs; En PLC-del som sköter alla signaler som skickas in och ut ur cellen; Slutligen en del som tar hand om robotar och automation i cellen.

En fabrik kommer att simuleras genom att bygga torn med klossar i olika färger. Byggprocessen börjar med att en operatör beställer ett torn från ett GUI. Operatören får instruktioner om hur klossarna till tornet skall placeras och förs därefter in i cellen. I cellen står robotar i väntan på att få bygga upp tornet med hjälp av instruktioner från den övergripande styrningen.

Arbetet har gjorts i samarbete med kandidatgruppen *SSYX02-16-01* som har skrivit en kompletterande rapport.

Ordlista

ABB	Företag som arbetar inom elkraft och industriautomation
Action	Handlingen i en operation
Backend	Den del av högnivån som skickar, behandlar och tar emot information
CPU	Central Processing Unit
DAG	Typ av graf som är acyklisk och riktad
FlexLink	Tyskt företag som tillverkar och säljer olika sorters monteringsband
Frontend	Användergränsnittet mellan högnivån och operatören
Graf	En samling noder som kopplas samman av kanter
GUI	Graphical User Interface
Högnivå	Samlingsnamn för de delar som berör den övergripande styrningen
I/O	Input/Output, samlingsnamn för enheter som hanterar datasignaler
Kant	Väg mellan två noder i en graf
KUKA	Tyskt företag som arbetar med automatiseringslösningar
Ladder	Ett logik-baserat kodningsspråk som används inom PLC-programmering
Nod	En punkt i en graf
Operation	Berskriver ett skede i ett schema för systemet
PLC	Progammable logic controller
Postcondition	Beskriver vilka krav som måste vara uppfyllda för att en Operation skall ses som avklarad
Precondition	Beskriver vilka krav som måste vara uppfyllda för att en Operation skall starta
RAPID	Kodspråk som används av ABB för programmering av industrirobotar
RFID	Radio frequency identification
Scala	Ett funktionellt och imperativt programmeringsspråk
Sequence Planner	Plattformen som högnivån är implementerad i
Service	Funktioner som implimenterats i Backend som kan anropas från Sequence Planer
SFC	Sequential Function Chart, ett kodspråk som används för att skapa rutiner i en PLC.
SOP	Sequence of Operations
State	Beskriver vilket läge som systemet beffiner sig i
SP-Atributes	En del av operationen som kan innehålla varierad information
TIA	Totally Integrated Automation, ett PLC programmeringsprogram från Siemens
Widget	GUI i Sequence Planner

Innehåll

Figurer

1 Introduktion

I dagens samhälle är det faktorer som datorkraft och informationsutbyte som driver på de tekniska förändringarna inom produktionstekniken [1]. Att sträva efter en kostnadseffektiv verksamhet är viktigt för att företag skall kunna hålla sig konkurrenskraftiga på marknaden. En stor kostnad för företag som bedriver produktion av fysiska produkter är att modifiera och byta produktionsätt. Denna process blir billigare om företaget har ett flexibelt produktionssystem som möjliggör snabba produktbyten, helst helt utan stopp [1]. Det kan därför vara av intresse för dessa företag att bygga så flexibla produktionssystem som möjligt. Kostnader kan hållas nere när marknadsefterfrågan fluktuerar, vilket är anledningen till att konceptet industri 4.0 håller på att utvecklas.

Begreppet industri 4.0 står för den fjärde industriella revolutionen. Termen användes först vid industrimässan i Hannover 2012 [3]. Efter ångmaskinen, elektriciteten och elektroniken går industrin in i en ny era där allt i produktionskedjan är smart och uppkopplat [4]. Målet är självorganiserade fabriker där varje produkt bär med sig information om hur den skall tillverkas och vem beställaren är. Eftersom maskinerna i tillverkningen också är uppkopplade ger det produkten möjlighet att kommunicera med tillverkningsutrustningen och tala om vilka steg den behöver gå igenom. Det som är målet med industri 4.0 system är massproducera individuella produkter på ett kostnadseffektivt sätt [4].

Det finns ett antal andra system som också strävar efter flexibilitet i produktionssystem. RMS, FMS och HMS är exempel på sådana. Ett RMS-system (Reconfigurable Manufacturing System) har möjligheten att producera olika varianter av produkter inom samma produktfamilj [1]. RMS-systemet inkluderar all mjukvara och hårdvara som arbetar, samt alla produktionskomponenter. Ett RMS-system kräver dessutom att komponenterna har hög integrerbarhet för framtida modifikationer. Integrerbarheten underlättar också när omställningar av produktionen skall göras. Komponenterna ska var enkla att diagnostisera för att man snabbt ska kunna upptäcka fel och brister. Sammantaget ska alla bitar i systemet ha bra kompatibilitet och möjlighet till förändring [1].

Ett annat system som dök upp under 1900-talet är FMS-systemet (Flexible Manufacturing System). FMS-systemet är ett tillverkningssystem i vilket det finns en viss grad av flexibilitet som gör att systemet kan reagera i händelse av både förväntade och oförutsedda förändringar [5]. Denna flexibilitet kan delas in i två kategorier: maskinflexibilitet och routingflexibilitet. Maskinflexibilitet omfattar systemets förmåga att ändras för att ta fram nya produkttyper och dess förmåga att ändra ordningen på de transaktioner som utförs på en produkt. Routingflexibilitet gör det möjligt att utföra samma operation genom olika maskiner. De främsta fördelarna med en FMS är dess höga flexibilitet i hanteringen av produktionsresurser som tid och ansträngning för att tillverka en ny produkt.

Det finns även HMS-system (Holonic Manufacturing System) som är ett koncept som kan användas för att uppnå flexibla tillverkningssystem.[6] HMS är ett av IMS-systemet (Intelligent Manufacturing Systems) sex större projekt som skapades till följd av en studie i början av 1990-talet. HMS-systemet föreslår ett antal nya steg inom tillverkningsindustrin som kan vårt gamla synsätt samtidigt sem den skapar många nya utmanande tekniska problem. Tanken bakom HMS är att tillhandahålla en dynamisk och decentraliserad tillverkningsprocess, där människor är effektivt integrerade så att ändringar kan göras dynamiskt och kontinuerligt.

Chalmers jobbar med att försöka beskriva produktionssystem med en mängd operationer. Ett sätt att beskriva en operation är med ett precondition, handling och ett postcondition. När precondition är uppfyllt startas en handling. Denna kommer utföras tills ett postcondition är uppfyllt [7]. Ett exempel på en operation kan vara om man ska gå igenom en dörr. Då är precondition att dörren skall vara öppen, handlingen är att passera dörren och postcondition är att personen i fråga har gått igenom dörren.

1.1 Syfte

Syftet med projektet var att utvärdera om det med hjälp av operationer går att modellera och bygga upp ett produktionssystem som kan producera unika produkter utan att behöva ställa om hela systemet. Ett mål var att få systemet i Chalmers PSL-labb (Production Systems Laboratory) att fungera.

1.2 Problembeskrivning

Projektet gick ut på att anpassa en produktionsanläggning som med operationer skulle kunna bygga produkter som definieras vid en beställning. Produktionsanläggningen finns i en av Chalmers labblokaler. Produkten kommer att vara ett torn av klossar i olika färger (gul, blå, grön och röd). Dessa kan kombineras i olika ordning och färger vilket ger högt antal möjliga unika kombinationer.

Produktionslinjen styrs i tre nivåer: högnivå, PLC samt robotstyrning. Dessa har olika uppgifter i produktionsanläggningen och det finns en kommunikation mellan dem. Detta har implementeras på ett generellt sätt så att nya funktioner senare skall kunna implementeras. Produktionsystemet, produktens flöde och kommunikationen beskrivs i kapitel 2.

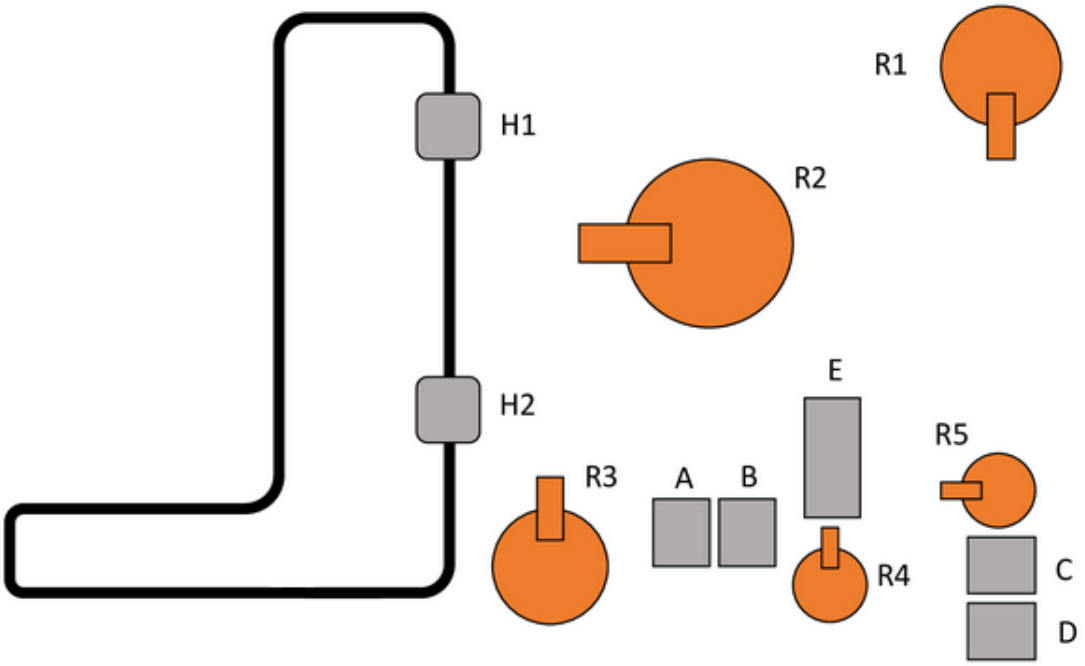
Problemställningar

I projektet togs ett antal frågor och problem upp.

- Hur ska man modellera och styra systemet med hjälp av operationer?
- Skriv kod sådan att den ger en tydlig överblick och blir återanvändbar.
- Hur ser ett tydligt gränssnitt för en operatör ut?
- Hur skall instruktioner presenteras för en operatör?
- Hur kan man optimera hämtning av klossar och konstruktion av produkten?
- Hur ska klossar sorteras för att minimera väntetider?
- Skall man hårdkoda alla produktvarianter eller skall koden byggas i takt med att nya modeller införs?
- Hur ska robotens verktyg se ut?
- Är flera robotar på samma produkt effektivare än en robot per produkt?

2 Produktionssystemet

Produktionssystemet sattes upp i PSL (Production System Laboratory) på Chalmers. I labbet finns en robotcell, se figur 1, där det finns 4 stycken robotar från ABB (R2 - R5) och en robot från KUKA (R1). Det finns också ett transportband från FlexLink med två hissar (H1 och H2). I projektet användes robotarna R2, R4 och R5, samt FlexLink-bandet och hissarna. Transportbandet och hissarna i cellen styrs av en PLC (Programmable Logic Controller). I cellen finns det även flera nöd- och säkerhetsstopp kopplade till en säkerhets-PLC, som stänger av strömförsörjning och tryckluft om någon av nödstoppen löser ut.



Figur 1: Översikt av robotcellens layout

Systemet startar när operatören matar in en order i ett GUI (Graphical User Interface) och trycker på beställningsknappen. Operatören får då instruktioner om hur klossar ska lastas på en palett. När lastningen är klar skickas paletten på FlexLink-banan till H1, där den lyfts upp och fixeras av en hiss. Sedan kommer R2 och flyttar paletten till någon av positionerna A-D. Klossarna flyttas sedan från paletten till position E av R4 och/eller R5. När paletterna är tomma flyttas de av R2 till position H2 där de sedan skickas tillbaka till operatören via FlexLink-bandet. När tornet är klart flyttas byggplattan som tornet står på till ett bord mellan H1 och H2. R2 förbereder sedan nästa order genom att ta en ny byggplatta och lägga på position E.

I figur 2 beskrivs kommunikationen i systemet. Operatören kommunicerar endast med frontend, den del av högnivån där bland annat GUI finns. Backend sköter kommunikationen till PLC:n som i sin tur kommunicerar med robotarna.

Figur 2: Översikt av kommunikationsordningen

3 Högnivå: En övergripande styrning

Produktionen startar med att ett klosstorn beställs. Denna beställnings behöver tas hand om och sekvenser av operationer som behöver utföras ska planeras. Det är högnivån som tar hand om detta.

Högnivån består av två delar: *frontend* och *backend*. Frontend är den del som tar in data från operatörens inmatningar och ger operatören nödvändig information. Den data som operatören matar in valideras och skickas sedan vidare till backend. I backend behandlas informationen och ny information skickas sedan vidare till PLC:n, se figur 2. Backend är en mycket beräkningstung del som innehåller optimeringsalgoritmer och funktioner som används för att skapa ritningarna för hur tornet skall byggas.

3.1 Programmering av styrningen

Backend programmeras i språket Scala som är ett imperativt funktionellt språk. Scala har fördelarna från både imperativa språk som Java och funktionella språk som Haskell [8]. För utveckling i Scala används IntelliJ som utvecklingsmiljö. Användargränssnittets menyer och knappar designas och programmeras med hjälp av AngularJS framework. För att utveckla gränssnittet kommer språk som JavaScript och HTML 5 att användas. Operationer och optimering kommer att programmeras i Scala och sedan visualiseras i Sequence Planner.

3.1.1 Sequence Planner

Sequence Planner är den plattform som har använts för web-interface, optimeringsalgoritmer, scheman och annan information som får systemet att fungera. Detta program är under utveckling och drivs av Chalmers. Programmet används för att modellera och simulera industrimiljöer för forskningsprojekt och utföra nödvändiga beräkningar för att få fram information om hur en produkt skall produceras. Sequence Planner är grundstenen som högnivån är implementerad i och även PLC-logiken har designats med hänsyn till att Sequence Planner toppstyr systemet. Web-interface kan användas för att kommunicera mellan Sequence Planner och en PLC som är uppkopplad till nätverket. När ett nytt projekt startas har detta nya förutsättningar och mål. Då krävs det också att nya web-interface, nya scheman och nya algoritmer utvecklas för att få systemet att fungera. Nya widgetar, services och algoritmer implementerats i antingen backend eller frontend. Widgetar blev implementerade i frontend och services samt algoritmer är implementerade i backend.

3.2 Användargränssnitt

Produktionssystemet bygger ett torn av klossar i ett flertal olika färger. När operatören har angivit hur produkten skall se ut i GUI:et, så kan operatören sedan trycka på en knapp för att skicka in ritningen på tornet till systemet. Vidare bearbetning av hur tornet skall byggas görs sedan i backend.

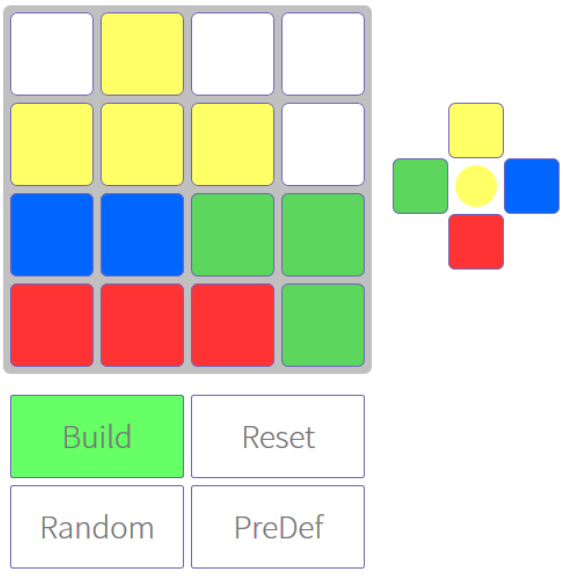
3.2.1 Omöjliga beställningar

När en beställning av en modulär produkt skall göras är det viktigt att systemet hindrar operatören från att göra felaktiga beställningar. Det kan finnas krav på valet av moduler eller den ordning som de placeras i. När klosstornet skall byggas är det viktigt att ta hänsyn till de begränsningar som finns. Exempelvis måste varje kloss placeras direkt på byggplattan eller ovanpå en annan kloss för att undvika att tornet kollapsar. Det krävs därför kontroll av varje beställning för att säkerställa att inga beställningar går igenom som inte är möjliga att genomföra.

Frontend avsvarar för att de beställningar som skickas till backend är möjliga att bygga. Om så inte är fallet ger frontend en varning till operatören och skickar ingen information till backend. När en beställning är godtagen och skickad till backend så beräknas sekvensen av operationer som skall utföras för att bygga tornet.

3.2.2 Beställningspanel

Beställningspanelen i figur 3 visar en grafisk representation av klosstornet. För att lägga till en ny kloss så väljer man en färg i färgväljaren till höger. Sedan placeras klossarna ut i tornet genom att trycka på de platser där klossarna skall placeras. När det önskade tornet är byggt så kan det beställas genom att trycka på `Build`. Knappen `Reset` rensar det som tidigare byggts och `Random` bygger ett slumpmässigt torn. Knappen `PreDef` tar fram ett fördefinierat torn.



Figur 3: Den panel operatören ser då ett nytt torn ska beställas.

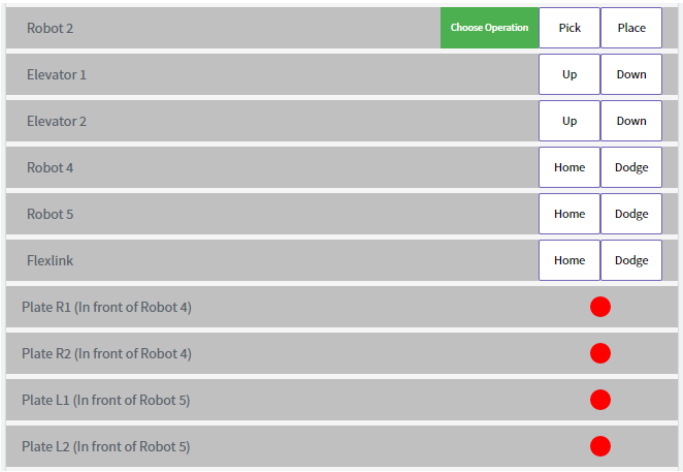
3.2.3 Operatörspanel

När en beställning har blivit godkänd så behöver klossarna som ska bygga upp tornet skickas in till systemet. Operatörspanelen i figur 4 visar hur klossarna skall placeras på paletten. Beroende på tornets form så kan det krävas två paletter för att få in alla klossar till systemet. Bilden visar tornets högra eller västra del då dessa behöver skickas in separat på olika paletter. Först visas den vänstra delen, när operatören har placerat klossarna och tryckt `tumme upp`-knappen så skickas den vänstra delen in och den högra delen ersätter den vänstra i operatörspanelen. Sedan upprepas proceduren för den högra delen och beställningen är klar.

Figur 4: Den panel som visar operatören hur klossarna skall placeras på paletten.

3.2.4 Återställningspanel

När problem inträffar kan det krävas en full återställning av produktionsscellen så att den kan köra igen. För att minska antalet återställningar finns en panel för manuell körning, där operatören kan rätta till fel utan att cellen behöver återställas. Den manuella styrningen skickar individuella operationers startsignaler till PLC för att styra robotarna. När PLC har utfört operationen och operatören har sett att operationen har utförts korrekt så kvitterar operatören att detta. Kvittering sker genom att sätta startsignalen till PLC:n låg igen vilket återställer operationen. ResetPanelen 5 innehåller också information om cellens viktigaste sensorer för att underlätta vid felsökningen. För att köra en operation väljer operatören manuellt läge. Sedan väljs den resurs som skall köras manuellt.



Figur 5: En panel för manuell styrning av cellen.

3.3 Schemaläggning

För att kunna uppnå effektivitet och flexibilitet i ett system måste produktionsflödet schemaläggas. Det finns otroligt många kombinationer om hur tornen kan utformas vilket gör det omöjligt att använda fördefinierade scheman för varje torn. Det krävs istället en modell av systemet som kan användas för att beskriva den ordning som saker och ting skall ske i. I modellen finns operationer, statevariabler och states representerat i Scala-kod. Operationer är enkelt beskrivet handlingar som utförs, statevariabler beskriver när saker kan ske och states är en samling av operationer och statevariabler. Dessa beskriver tillsammans det läge systemet befinner sig i.

Givet ett specifikt state kan möjliga operationsval utformas och en operation kan väljas som leder till ett nytt state. Genom att låta noder representeras av states och kanter representeras av operationer kan de två delarna tillsammans representera en graf. Genom att använda grafsökningsalgoritmer kan sedan ett tidsoptimerat schema tas fram. Schemat kommer vara optimerat med avsende på den minsta totala verksamma tiden och är en typ av Gantt-schema. När grafsökningen är klar finns information i grafen som beskriver när alla operationer startar och i vilken ordning. Den informationen kan sedan användas och kodas om till ett schema som kan köras av produktionssystemet.

3.3.1 Operationer & SOP

En operation, betraktad från schemaläggningperspektivet, består av tre delar och tre faser. Delarna är precondition, action och postcondition och faserna är initialfas, handlingsfas och slutfas. I början befinner sig operationen i initialfasen och väntar på att precondition skall bli uppfyllda. När dessa blivit infriade går operationen in i handlingsfasen där action utförs. Action avslutas med att systemet signalerar att action är klar eller att postcondition är uppfyllda, varpå operationen går in i slutfasen. Ett exempel på en operation kan vara att en robot skall flytta ett kloss från en position till en annan. Precondition för denna operation är då bland annat att roboten inte har en annan uppgift och andra robotar inte blockerar resvägen.

Det action som utförs är då att flytta klossen och operationen ses som avklarad när roboten utfört denna uppgift [7].

Varje operation innehåller någon from av action. Om den action som skall utföras är fysisk måste det också finnas en eller flera enheter i systemet som kan utföra dem. Dessa kan till exempel vara robotar och definieras som systemresurser. I detta fall är systemresurser produktiva delar av systemet som till exempel robotar, maskiner eller människor. Det finns även operationer som endast påverkar statevariabler och har ingen fysisk representation. [7].

Systemet har modellerats i Sequence Planner, med operationer som har verklig representation och sådana som endast påverkar statevariabler. Operationerna är uppbyggda av namn; precondition och action; postcondition och action samt SP-Attributes. SP-Attributes innehåller information som behövs för att starta operationen i verkligheten eller den teoretiska tiden som operationen tar. De systemresurser som finns är två robotar som bygger, en större robot som flyttar paletterna, två hissar och en operatör. Alla dessa resurser är kopplade till operationer som har fysisk representation.

De två små byggrobotarnas operationer handlar om att flytta klossarna från de olika paletterna till byggplattan. Denna rörelse är uppdelad i två operationer; En operation som plockar upp och en operation som lämnar. Det finns även ett antal statevariabler som bestämmer vilka plocka- och lämnaoperationer som får starta. Ett exempel är den operation som lämnar en kloss. För att en kloss ska kunna lämnas krävs det att den placeras direkt på plattan eller ovanpå en annan kloss. De små robotarna kan också placera sig i ett läge som kallas ”Dodge”, vilket är nödvändigt för att den stora roboten skall placera klosspaletter. När de små robotarna utför en operation så utförs två action. Den ena sker i början av operationen och denna action representerar att klossen plockats upp eller lämnats och roboten i fråga bokas. Den andra action sker när operationen är fullföljd och då avbokas roboten.

Den stora robotens operationer går ut på att flytta de olika paletterna till och från byggplatsen. Operationerna är specifika i den mening att systemet vet på vilken plats som de olika paletterna finns. Denna resurs har även operationer som ställer in och plockar ut byggplattor. De två hissarna som används vid in- och utkörning av klosspaletterna har också operationer som modellerar dessas action. Hissarna kan höjas och sänkas och även dessa är specifika i den mening att systemet vet vad som hissas upp. Den sista resursen som är modellerad är operatören. De operationer som är kopplat till denna är att skicka in klosspaletter och byggpaletter in i systemet. Sedan finns det en del operationer som endast ändrar statevariabler. Dessa är de operationer som initierar det första statet. Dessa operationer väljer vilka klossar som skall placeras, om det behövs en eller två klosspaletter samt den initieringsoperation som ger de de andra nödvändiga statevariabler startvärd. De flesta får värdet false med undantag för init-variabeln. Det finns även operationer som signalerar att en rad av, eller hela, tornet är klart. Till sist finns ett set av operationer som används i initieringsfasen för att bestämma vilka klossar som skall placeras var.

Flera operationer som körs i en sekvens eller parallellt kallas för SOP (Sequence of Operations). En SOP är ett körbart schema och om det skulle representeras grafiskt skulle de likna ett flödesschema eller Gantt-schema. I en SOP finns det sekvensiella delar och även parallella delar, vilket innebär att sekvenserna i vissa steg kan vara parallellt körbara [7].

Tanken var att det skulle skapas nya SOP:ar allt eftersom operatören beställer nya torn, men denna funktion blev inte implementerad i det slutliga systemet. Varje beställning skulle undersökas och en SOP skulle tas fram för varje specifik beställning. En SOP tas fram på så vis att efter en grafsökning finns startordningen för alla operationer, vid vilken tidpunkt i produktionen som operationerna startar. Denna information kan tolkas som ett Gantt-schema. Sedan kan detta Gantt-schema kodas om till en SOP.

3.3.2 States & State-variabler

Ett state indikerar vilket läge som systemet befinner sig i. De delar som bygger upp ett state är operationer och statevariabler. Det är med hjälp av statevariablerna som man kan avgöra vilka operationer som kan köras i ett givet state. När man talar om systemets läge menas alltså vilka operationer sker, har skett och ska ske. Den andra biten är vilka statevariabler som är sanna respektive falska. Genom att analysera båda delar samtidigt kan man få fram mycket information om systemet. En sådan analys görs från ett teoretiskt perspektiv där inga avbrott eller förseningar sker. På det här sättet kan man beskriva är hur lång tid som systemet har kört, vilka delar av produktionen som är klara, vad som finns kvar att göra samt hur långa tid detta skulle ta [7].

States används för att modulera vilket läge som systemet befinner sig i. För att kunna grafsöka behövs ett state att utgå ifrån för att hitta operationer som kan starts. När de startbara operationerna är kända kan någon av dem väljas för att skapa ett nytt state med uppdaterade state-variabler. För varje specifikt torn kommer ett initialstate att skapas och när nästan alla operationer i statet har utförts och tornet är klart har man nått ett endstate. Anledningen till att inte alla operationer utförs är att initialstatet skapas med för många operationer. Alla operationer som behövs för att bygga alla typer av torn finns implementerade vilket leder till att det implementerades en avslutningsoperation som signalerar att tornet är färdigbyggt. Den mer naturliga avslutningen hade varit att alla operationer har körts. Anledningen till att man har med alla möjliga operationer är att optimeringen av systemet hade påverkats negativt om man utlämnar operationer från initialstate. Ett exempel är den stora roboten och dess operationer som placerar klosspaletterna. Det går fortare om den ställer klossar hos båda de små robotar så att båda kan jobba istället för att en liten robot gör allt arbete själv.Om operationer då utelämnas ur initial-

statet som hade tillåtit detta hade det möjliga schemat försvunnit. Vilket innebär att de initialstate som skapas som har flera möjliga produktionsvägar kommer innehålla operationer som inte körs. Vilket ledde till att schemats slut identifieras i grafsökningen.

Statevariabler beskriver läget som systemets resurser befinner sig i. Exempelvis så har systemets hissar flera variabler kopplade till sig. Det finns bland annat variabler som bestämmer hissarnas lägen, dessa beskriver om hissen är uppe eller nere. Finns det något som kan bli upphissat och vilken typ som det skulle handla om beskrivs detta också utav dessa variabler [7].

De statevariabler som har implementerats behandlar systemets olika resurser och tillstånden för de olika paletterna. De variabler som är kopplade till robotarna beskriver om de är verksamma, om de befinner sig i kända positioner eller om robotorna håller i klossar. Eftersom systemet endast kan bygga torn i en nivå i taget finns det även variabler som beskriver om nivån under är färdigbyggd. Sedan kan det förekomma att vissa kolumner skall vara tomma, men då ses de positionerna som färdiga. Operatörsvariablerna beskriver vilka paletter som operatören har skickat in. En extra viktig statevariabel är den som är säger till när tornet är färdigbyggt. Denna variabel är inte direkt kopplad till en resurs utan till byggplattan. Det finns även variabler som beskriver vad de olika paletterna innehåller. Byggplattans innehåll och vad som skall placeras finns också beskrivet. Positionerna där klosspaletter eller byggplattor placeras finns representerat bland dessa variabler.

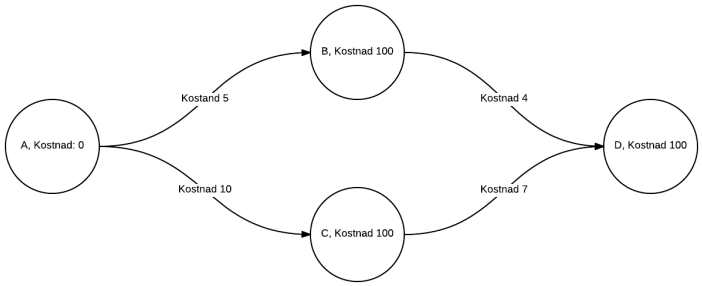
3.3.3 Grafsökning & Optimering

En samling av noder och kanter kallas för graf. Noderna kopplas samman av kanterna och kanterna kan ha vikt och riktning. Om en riktad kant kopplar samman två noder kan man bara ta sig mellan dessa i kantens riktning. Om kanten har en vikt så innebär det att det uppstår en kostnad när denna används vid färd mellan två noder. Dessa kostnader kan exempelvis vara avstånd eller tid. Den typ av graf som används i systemet kallas för riktad acyklisk graf eller DAG på engelska. Det som är specifikt för en acyklisk graf är att det inte finns några cykler i grafen. En cykel i en graf innebär att man kan vandra i en riktning i grafen och nå en tidigare besökt nod. Att grafen är riktad innebär att man endast kan färdas i samma riktning som kanten. [9].

Systemet har beskrivits som en riktad acyklisk graf där noderna är states och kanterna är operationer. Detta medför att grafen kan bli väldigt stor eftersom det finns många möjliga vägar att bygga upp tornen på. En annan faktor som bidrar till den potentiella storleken av grafen är det faktum att två noder kan innehålla samma state. Noderna är i så fall lika i den mening att de statevariabler och operationer som utförts är samma, men att operationerna har exekverats i olika ordning. En möjlighet hade varit identifiera sådana states och riktat om kanterna så att de pekar på samma nod. Det kan påverka kostanden att nå noden vilket hade behövt uppdateras men de hade ej varit svårt att implementera.Dessa faktorer innebär det

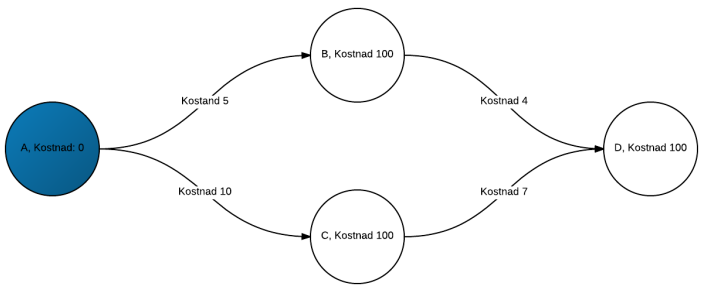
skulle bli en minnestung och onödig uppgift att spänna upp hela grafen. Istället skapas det nya noder allt eftersom algoritmen utförskar vilka noder som är möjliga att nå, vilket resulterar i att inte hela grafen utförskas.

Dijkstras algoritm kan användas för att hitta kortaste vägen till en slutnod D i en viktad graf, som kan vara riktad eller oriktad, så länge som det finns en initailnod A; se figur 6. I algoritmen är K_{AB} kostnaden mellan noderna A och B, och K_A är kostnaden för A.



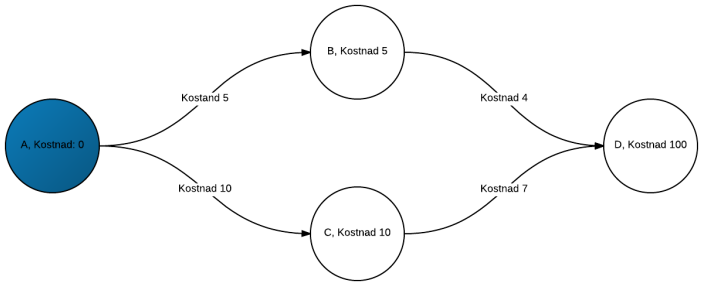
Algoritmen kommer först att addera initalnoden till ett set som innehåller utforskade noder och uppdatera A: $K_A = 0$.

Figur 6: Initialläge



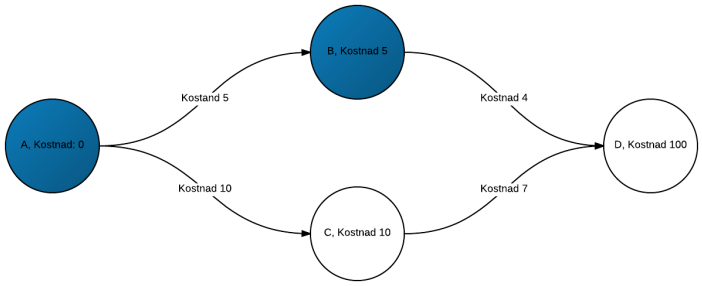
Sedan kommer alla närliggande noder, B och C, adderas till setet av outforskade noder.

Figur 7: Initialnoden utforskad



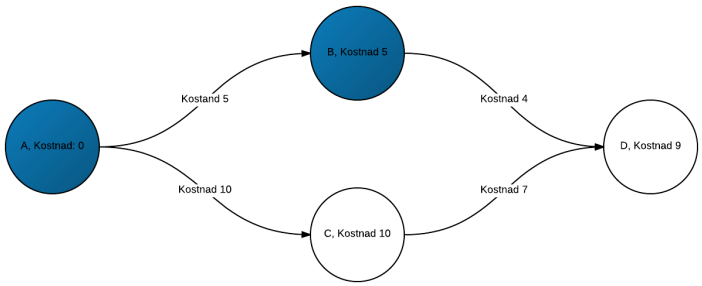
Figur 8: Närliggande noder updaterade

Efter detta uppdateras alla närliggande noders kostnad. Noduppdateringen sker genom att man tittar på kravet $K_A + K_{AB} < K_B$. Om kravet uppfylls gör man $K_B = K_A + K_{AB}$. Alla noders kostnad är initialt stor, $K_B = \infty$ och $K_C = \infty$, vilket leder till att de nya nodvärdena summeras som kantvärdet plus nodvärdet som kopplar ihop noderna: $K_B = K_A + K_{AB} = 0 + 5 = 5$, $K_C = K_A + K_{AC} = 0 + 10 = 10$.



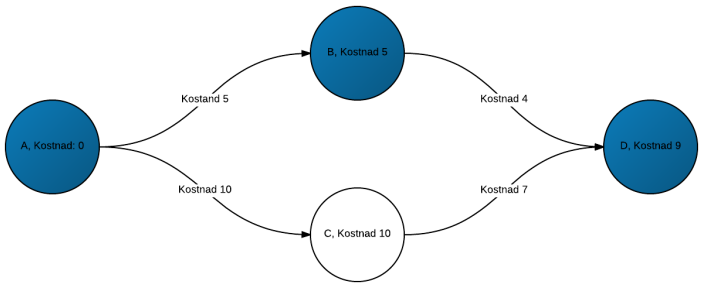
Figur 9: Outforskad nod med lägst kostnad vald

Sedan kommer den nod som här lägst kostnad att utforskas.



Figur 10: Närliggande noder uppdaterade

I nästa iteration kommer den närliggande noden, D, undersökas och dess nya kostnad kommer uppdateras: $K_D = K_B + K_{BD} = 5 + 4 = 9$.



Sedan utforskas den nod med lägst kostnad i det set som inte har utforskats.

Figur 11: Outforskad nod med lägst kostnad vald vilket avslutar sökning då sökt nod funnen

Denna process upprepas sedan tills den sökta noden, i detta fall *D*, är funnen.[10].

Idén var att implementera en sökalgoritm som söker igenom alla states (noder) och operationer (kanter) och hittar den snabbaste vägen. Sökningen skulle generera ett tidsoptimerat Gantt-schema som sedan skulle kunna översättas till en körbar sekvens av operationer. Sökalgoritmen blev inte helt färdig utan det som blev klart var systemmodellen (operationerna och statevariabler). Tanken med sökalgoritmen var att den skulle utgå från Dijkstras algoritm. Samma liststruktur med besökta och obesökta noder skulle användas, resekostnaden till varje nod skulle öka successivt och den mest effektiva vägen skulle väljas. Det som skiljer den påbörjade algoritmen och en standard Dijkstras algoritm är att det kan köras flera operationer samtidigt. På det här sättet kan nå målet fortare eftersom att man startar flera operationer samtidigt.

Eftersom varje operation har tre faser innebär det att det krävs tre states för att representera operationens alla lägen. Från första noden kan man starta operationen, då går man in i startfasen. Kostnaden för den kanten är 0. När detta läge är nått så har de action som är associerade med precondition körts. Om det är en operation som körs som är kopplad till en liten robot så innefattar det bokning och klosshantering. Sedan kan en ny kant följas och operationen köras. Operationen ses då som avklarad och kantens vikt är då operationens körningstid. När operationen är avklarad sker också den action som är kopplad till postcondition. I fallet med den lilla roboten skulle de vara att avboka roboten. Sedan skulle den sista nodens resekostnad öka genom att addera körningstiden och kostanden som förgående nod hade. När två operationer kan startas samtidigt kan det ske och det skulle minska den totala resekostnaden då flera operationer exekveras i samma kant. Sedan skulle operationerna köras samtidigt. Om operationerna hade haft olika körtider så hade den med kortast körtid avslutats först. Sedan hade nästa state nåtts och om det är möjligt hade nya operationer startats. Körtiden för den operationen som ej blev avslutad hade subtraherats med körtiden som den avslutade parallellakörda operationen hade.

Ett moment är att slutet på schemat skall hittas. Detta kommer att göras genom att identifiera ett state i grafsökningen som det inte kan startas några operationer från. Det är samma princip som att alla operationer behövs köras för att bygga klart tornet är färdiga. Dock får det en annorlunda implementation i koden. Den enklaste implementationen hade varit att identifiera när alla operationer är klarkörda. Dock är detta inte möjligt och det beror på att de operationer som behövs köras kan ändras från torn till torn.

4 PLC: En logisk styrenhet

För att styra maskiner och robotar ute i industrin så används ofta en PLC (Programmable Logic Controller). Detta görs även i det produktionssystemet som byggdes upp i detta projekt. I detta kapitel så introduceras först en del av teorin bakom en PLC, hur den fungerar och arbetar, följt av hur PLC:n användes i projektet.

4.1 PLC:ns Uppbyggnad

Termen *logisk* controller används för att den till största delen behandlar logiska beräkningar [12]. Från början skapades PLC:n för att ersätta relä- och timersystem [11]. Jämfört med relä-systemen så gör PLC:n samma uppgift men på ett mer flexibelt sätt. Till exempel går det att förändra funktioner, genom att använda in- och utgångar på andra ställen i koden, utan att behöva koppla om kablar som man behövde göra mellan reläerna.

En PLC består av flera delar: en CPU (Central Processing Unit), gränssnitt för kommunikation samt in- och utgångar, gränssnitt, en programmeringsenhet, strömförsörjning och en minnesenhet. [12].

En CPU innehåller en eller flera microprocessorer som läser insignaler och beräknar hur utsignaler ska sättas med hjälp av programmet som ligger i minnesenheten. CPU:n har även rutiner för att se att PLC:n fungerar som den ska och för att kommunicera med andra enheter.

Gränssnittet för in- och utgångar används för att kommunicera med systemet. Ingångar läser givare eller knapptryckningar i systemet som styrs och utgångar skickar signaler till delar som motorer, reläer, magnetventiler och liknande. In- och utgångarna kan hantera antingen digitala eller analoga signaler. Gränssnittet behandlar all data och omvandlar den så att CPU:n kan hantera den.

I många system så används flera PLC:er och de behöver då kommunicera med varandra. Med kommunikationsgränssnittet kan koppla PLC:n till ett nätverk med andra enheter som kan prata med varandra. I ett sådant nätverk kan gränssnittet verifiera andra enheter, samla in data, sköta synkronisering och upprätthålla uppkoppling.

Programmeringsenheten behövs för att kunna skriva ett program som sedan överförs till minnet. Denna enhet kan vara en liten handhållen enhet som har en display och knappar för inmatning, eller en terminal med tangentbord. I många fall idag så används en PC, ofta en laptop, med gränssnitt för att kunna koppla in sig på PLC:n och föra över programmet till den.

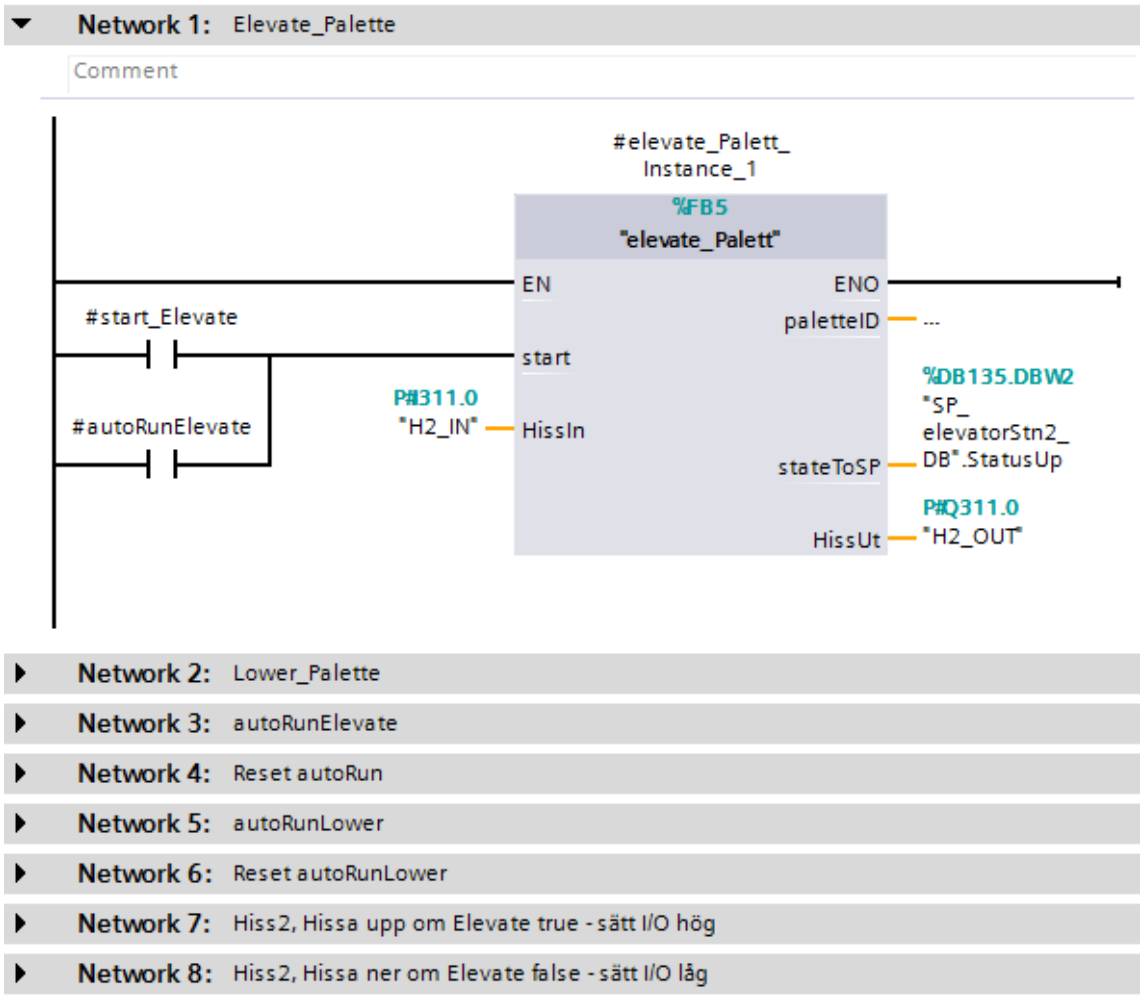
Metoden att programmera en PLC är designad så att det ska vara lätt för att användarna att skriva och läsa koden, eftersom arbetare inom industrin inte alltid har stora kunskaper om programmering [12]. Denna metod kallas ladder-programering och är ett av de vanligaste metoderna att programmera en PLC på [11]. Ladder baseras på ”ladder-diagram” som användes för att beskriva funktionen av system med reläer [13]. Att programmera en PLC var alltså i princip samma sak som att rita ett reläschema, vilket man kunde i industrin. Detta gjorde det alltså möjligt för de som inte är insatta i programmering att skriva kod till en PLC [12]. Det finns också fler metoder att programmera en PLC [11]. En annan metod som man använda kallas för SFC (sequential Function Chart) Till skillnad från ladder-kod så arbetar en SFC sekvensiellt likt ett flödesschema [14].

4.2 Styrning av cellen

PLC:n har till uppgift att utföra operationer och kontrollera givare i cellen. Dessa operationer är både sådana som PLC:n startar direkt och sådana som den säger åt robotarna att starta. Vissa operationer har startsignaler kopplade till högnivån samtidigt som andra styrs sekvensiellt av PLC:n. PLC:n är kopplad till det lokala nätverket i PSL och kan därmed kommunicera med Sequence Planner. Koppling till Flexlinkbandet sker via I/O-rack och robotarna via Profibus. Givare kontrolleras för att veta var paletter och byggplattor finns och vilka lägen hissarna är i. Det finns signaler från en säkerhets-PLC för att kontrollera säkerhetsstopp, som dörr till cellen eller ljusbommar, och nödstopp.

4.2.1 Kodstruktur

PLC-programmet är strukturerat så att de enklaste momenten är så generella som möjligt och kan användas av olika operationer på olika nivåer i programmet. Operationerna är grupperade under resurser i produktionssystemet och är kodade i ladder för att då systemets egenskaper blir lätta att avläsa. I cellen finns det två hissar på FlexLink-bandet och dessa behöver kunna lyfta upp och sänka ner paletthållarna. De två hissarna definieras som två olika resurser. Den enskilda hissresursen är ett funktionsblock som innehåller alla dess operationer, i detta fall `LiftPalette` och `LowerPalette`. Resursblocken för de två hissarna är nästan identiska fast med skillnaden att de har olika in- och utdata.



Figur 12: Exempel på en resurs, i detta fall `Elevator_Stn2` som innehåller alla abilities Hisstation 2 kan utföra.

4.2.2 Datatyper

För att autogenerera kod krävs det att alla invariabler sätts samman i en datatyp och alla utvariabler i en annan. Att ordna variabler i datatyper ger en lättöverskådlig och konsekvent struktur som underlättar kodning samt kommunikationen mellan PLC och Sequence Planner. Datatypers interna variabler har en konsekvent struktur, se figur 13. Varje gång en ny instans av samma datatyp skapas, måste också input/output kopplas in i samma följd som datatypens interna variablers offset. Om PLC-taggarna är i oordning så måste nya variabler skapas i rätt minnesordning. De gamla variablerna måste då manuellt kopplas med hjälp av kod till nya variablerna som sedan kopplas till datatypen. För att skapa en ny resurs av samma typ kopieras hela resursblocket men input/output till blocket ändras till det nya input/output, se figur 12. Dessa är av samma datatyper som den kopierade resursens), se figur 13 där `H2_IN` av datatypen `in_elevatorStn` används.

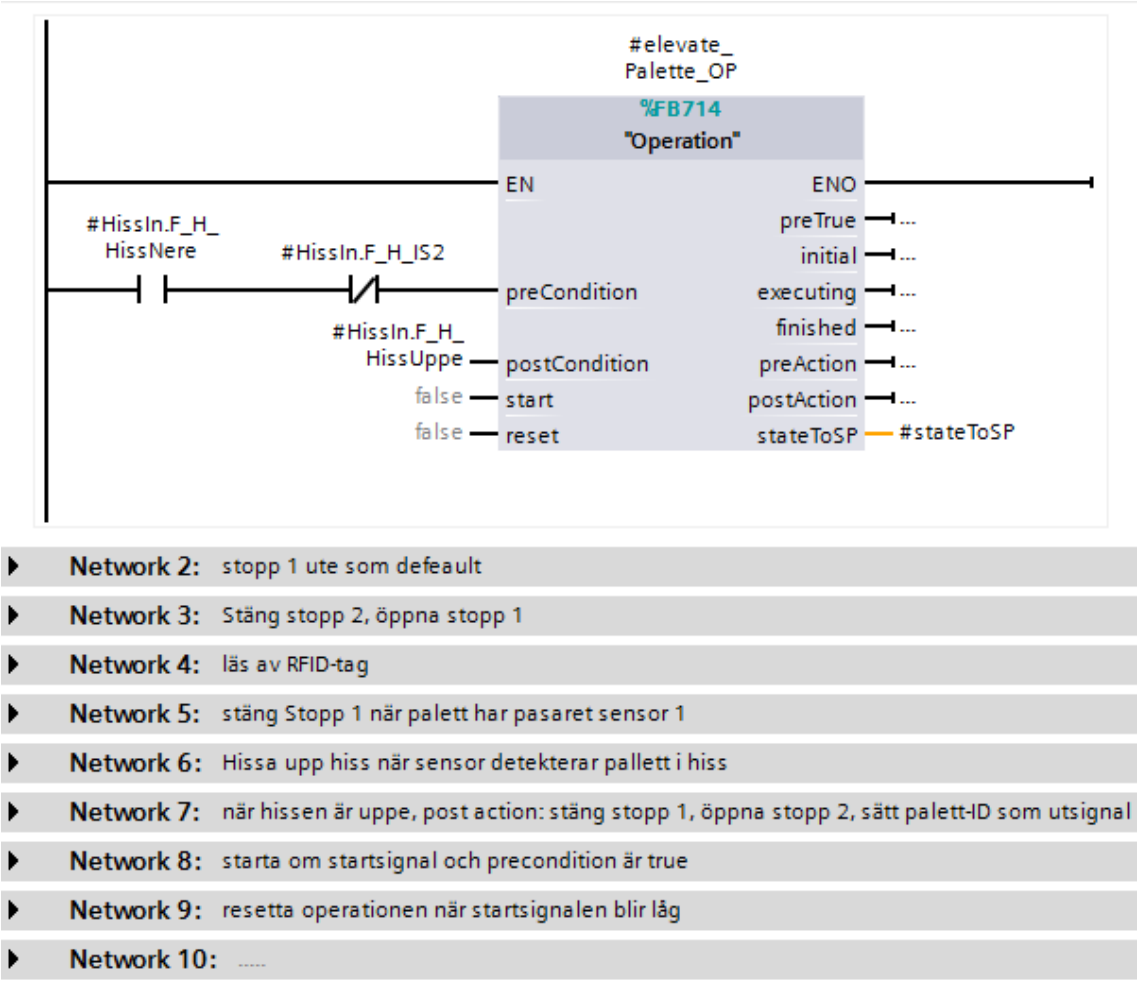
1150	H3_IN	Flexlink Kand16	"IN_elevatorStn"	%I312.0
1151	F_H_IS1		Bool	%I312.0
1152	F_H_IS2		Bool	%I312.1
1153	F_H_HissNere		Bool	%I312.2
1154	F_H_HissUppe		Bool	%I312.3
1155	F_H_IS3		Bool	%I312.4
1156	F_H_spare0		Bool	%I312.5
1157	F_H_spare1		Bool	%I312.6
1158	H2_OUT	Flexlink Kand16	"OUT_elevatorS...	%Q311.0
1159	H2_IN	Flexlink Kand16	"IN_elevatorStn"	%I311.0
1160	F_H_IS1		Bool	%I311.0
1161	F_H_IS2		Bool	%I311.1
1162	F_H_HissNere		Bool	%I311.2
1163	F_H_HissUppe		Bool	%I311.3
1164	F_H_IS3		Bool	%I311.4
1165	F_H_spare0		Bool	%I311.5
1166	F_H_spare1		Bool	%I311.6

Figur 13: Exempel av två variabler av datatypen `IN_elevatorStn` som innehåller all indata från en hisstation.

4.2.3 Styrning av FlexLink-bandet

PLC:n styr FlexLink-bandet och tillhörande resurser såsom hissarna direkt via PLC:ns I/O-portar. Boolska variabler är kopplade till in- och utportarna som i sin tur är kopplade till en I/O-modul vid FlexLink-bandet med bistabila mangnetventiler som med hjälp av tryckluft driver och hissar och stopp. När operatören placerat klossar på en palett lägger han den på en fixtur som har en RFID-tag (Radio Frequency IDentification) med ett nummer på. Högnivå säger till var paletten skall placeras i cellen, operatören lastar paletten och kvitterar att det är gjort. Paletten skickas då till roboten med operationen `PaletteToRobot` och direkt efter att paletten skickas iväg på bandet mot Hiss 2 så läses dess RFID-tag av och fixturens ID-nummer kopplas i en array med alla fixturer till positionen den skall läggas på. När paletten når resursen *Hisstation 2* som kör operationen `elevate_Palette`, se figur 14, som läser av palettens ID-tag.

Paletten passerar sedan induktionssensor 1 som ligger direkt efter stopp 1 som då stängs, så endast en palett passerar. Paletten fortsätter in i hissen vars stopp är aktiverat och när hissens sensor känner av paletten startas upphissningen. När hissen höjts är postcondition uppfyllt, stopp 2 öppnas och palettens ID sätts som utsignal `paletteID` från operationen, se figur reffig:Resurs. Palettens ID används för att ta fram den position roboten skall lyfta paletten till, och `PaletteID` skickas vidare som index till en array enligt `PaletteNrToPos[PaletteID]` och palettens position används som parameter till R2s operation som hämtar paletten från Hissen.



Figur 14: Operationen elevate_palette som körs automatiskt av PLC:n när ett torn beställts. elevate_Pal

4.2.4 Styrning av robotarna

Kommunikation mellan PLC och Robot sker via programnummer. Alla robotrörelser är definierade som operationer i PLC:n. Precondition samt postcondition är att roboten befinner sig i start- respektive slutpositionen. När precondition är uppfyllt samt en startsignal sätts hög så startar operationen i PLC:n. Då skickar PLC:n programnummer samt eventuella parametrar korresponderande till specifika rörelser. När robot-controllern tar emot programnummret sker en handskakning mellan robot och PLC genom att roboten skickar tillbaka samma programnummer den fick in. När handskakningen är godkänd skickar PLC:n startsignalen `cykelstart` till roboten. Det är även ett postcondition att roboten utfört eventuella uppgifter som att greppa en kloss. Parametrar används för att hämta eller sätta ut en kloss på en specifik plats utan att behöva skapa en helt ny operation med unikt programnummer. Mer information finns i kapitel 5.2.2.

4.2.5 Kommunikation med Sequence Planner

Om en resurs skall användas av Sequence Planner kopplas dess operationers startvariabler samt status till ett datablock som i figur 15. Variablerna i datablocken ligger PLC:ns minne och kan både läsas och skrivas från Sequence Planner. Sequence Planner kan därmed läsa operationernas status genom `subscribe`. En Operation kan befinna sig i 4 olika lägen. Om den är `init` så befinner sig operationen i init men precondition är inte uppfyllt. `Ready` betyder att operationen är i init, precondition är uppfyllt och därmed så startar operationen när startsignalen sätts hög. Generellt så väntar inte högnivån tills att operationen är redo utan försöker att starta operationen så fort den är i init. PLC startar sedan operationen när precondition blivit uppfyllt. Operationens tredje state är `working` som betyder att den har startat men postcondition ej uppfyllt. `Done` innebär att postcondition är uppfyllt, operationen är då klar men ej redo för omstart. När Sequence Planner läser av att operationen är klar, återställer Sequence Planner operationen genom att sätta startsignalen låg.

SP_elevatorStn2_DB

		Name	Data type	Offset
1		Static		
2		StartUp	Bool	0.0
3		StartDown	Bool	0.1
4		StatusUp	Int	2.0
5		StatusDown	Int	4.0

Figur 15: Datablocket för Sequence Planners styrning av resursen `elevatorStn2`.

4.2.6 Zoner och Bokningssystem

Högnivån sätter startsignaler till robotarna höga när den har beräknat vilka operationer som skall utföras, men utan hänsyn till alla preconditions som finns i PLC:n. PLC:n startar sedan robotarna när dess precondition är uppfyllda. Högnivå och PLC kan köra robotoperationer parallellt. Flera robotar kan utföra olika operationer samtidigt. Då robotarna inte kan avgöra av om det finns hinder i vägen för deras rörelser måste robotarna hindras från att köra in i varandra. Zoner där flera robotar kan befinna sig i samtidigt och därmed kollidera kallas *kritiska zoner*. När roboten skall färdas in i en sådan kritiskt zon skickar roboten en *byte*, i koden kallad `ZoneRequest`. `ZoneRequests` värde korresponderar till zonen roboten skall in i. Sedan väntar roboten på att PLC:n skickar en boolsk variabel, `ZoneRequestGranted`, som säger att den har tillåtelse att gå in i zonen. När en robot skickar `ZoneRequest` till PLC:n läggs då roboten i en kö för zonen den vill in i. När zonen är tom skickar PLC:n `ZoneRequestGranted` hög till den robot längst fram i kön. För att R2 ska in i vissa zoner krävs det att R4 och R5 har flyttat sig ur vägen. I dessa fall lägger sig R2 i kö i flera zoner samtidigt och när R2 har alla zoner bokade så får de andra robotarna flytta sig ur vägen. Detta sker genom operationen `goToDodge` och först när R4 och R5 skickat tillbaka att de är i `Dodge` kan R2 köra.

5 Robot: En resurs i systemet

Industrirobotar används ute i produktionssytem till tunga samt repetativa arbeten, där sekvenserna är oftast hårdkodade [15]. Produktionsystemet i PSL efterliknar systemen ute i industrin och har samma begränsningar för flexibilitet. I cellen finns det fem stycken robotar. Fyra av dem är från ABB och en är från KUKA. I arbete användes en av ABB robotarna (R2) för att flytta material in och ut ur cellen, och två ABB-robotarna (R4, R5) används för att bygga tornen. Robotarna programmeras i ABB:s egna programspråk RAPID som introduceras i kapitel 5.1.

5.1 Introduktion till RAPID

En robots rörelse går mellan två punkter i rummet, s.k. `robtargets`. En `robtarget` innehåller både en position i rummet samt robotens ledvridningar, dvs. hur robotens olika leder skall stå vid den givna punkten. När en rörelse skapas, kopplas två eller flera `robtarget` ihop med `Move`-instruktioner. De två vanligaste `Move`-instruktionerna är `MoveJ` (Joint) och `MoveL` (Linear). En `MoveJ` tar sig mellan två punkter i en icke-linjär rörelse och en `MoveL` medför en linjär rörelse mellan två punkter samtidigt som verktygets orientering förblir fixt relativt robotens bas.

Vilken `Move`-instruktion som är bäst att använda beror på vilka positioner roboten skall röra sig emellan och hur dess arbetsområde ser ut. Vid stora ytor och långa avstånd mellan punkterna fungerar `MoveJ` bäst eftersom att roboten då utför den rörelse som för lederna tar kortas tid [17]. Fördelen är att man bara behöver två punkter för att skapa en rörelse, men man kan inte vara helt säker på hur den kommer att se ut. För att minska risken att roboten kolliderar med något kan man låta den köra längs ett antal ”viapunkter”.

Om man vill ha mer kontroll över robotens bana kan man istället använda sig av en `MoveL`-instruktion. `MoveL` håller också verktyget plant under translaterande rörelser. En begränsning som finns i `MoveL` är ingen av robotens leder får vrida sig mer än 90° under en rörelse mellan två punkter. Denna begränsning finns inte hos `MoveJ`.

En Robot kan också kopplas in på en I/O-buss för kunna kommunicera med till exempel en PLC. En `DI` (Digital Input) signal kan komma från t.ex. en PLC eller från en annan robot och kan till exempel användas som en startsignal. En `DO` (Digital Output) är istället utsignal från roboten och sätts med en `SetDO`-instruktion. `SetDO`-instruktionen kan till exempel användas för att öppna och stänga verktyg eller skicka information till en PLC.

Rörelser och `SetDO`-instruktioner kan kopplas samman i en `Procedure`, förkortat `PROC`. En `Procedure` kan vara att flytta roboten från A till B, använda ett verktyg vid punkt C eller nollställa en signal till PLC:n. `Procedures` samlas i sin tur ihop i `Modules`. Man kan då skapa `Modules` som har hand om specifika operationer, t.ex. signalbehandling eller enklare beräkningar. Det finns alltid en main-modul då det är den som startas när roboten sätts i autoläge.

En ABB robot kan programmeras på två sätt: *Online* och *Offline*. Online betyder att man arbetar med robotens styrenhet (FlexPendant) för att styra (jogga) roboten till olika platser. Man kan sedan spara ner den punkt roboten befinner sig i som ett `robtarget`. Dessa punkter kan man antingen föra in i en `Procedure` direkt, eller så kan man spara dem till senare.

Offline-programmering görs vid vid en dator och utför programmeringen i en simulerad miljö. På samma sätt som i Online-programmeringen så körs den simulerade roboten för hand till de punkter som skall sparas som `Robtargets`. Offline-programmering är generellt sett snabbare då kritiska zoner där risk för kollision lätt identifieras. Om robotar kolliderar under simulering kan orsaken till kollisionen åtgärdas utan risk att skada det fysiska systemet. Nackdelen är att man behöver ha en noggran 3D-modell av arbetsområdet för att vara säker på att rörelserna sedan blir identiska i den fysiska cellen.

5.2 Programmering av robotar

För att kunna bygga ett torn måste robotarna ha olika `Procedures`. R2 måste ha `Procedures` för att hämta och lämna både byggplattor och paletter; R4 och R5 har istället `Procedures` för att hämta klossar från paletterna och lämna dem på byggplattan. Vidare så måste R4 och R5 kunna vika undan när R2 skall arbeta i närheten av dem. Alla robotarna har ett antal hempositioner med tillhörande utsignaler som sätts när roboten befinner sig vid dem. Alla banor börjar och slutar i en hemposition och signalerna används för att säga till PLC:n var i cellen robotarna befinner sig.

Varje `Procedure` börjar med att roboten sätter alla hempositioner till `FALSE`. Därefter åker roboten till en specifik destination och utför ett uppdrag. Om robotarna använder sina verktyg sätts även en utsignal till PLC:n som säger om verktyget är aktiverat eller inte. Slutligen åker roboten tillbaka till en hemposition och sätter tillhörande utsignal till `TRUE`. För att göra kommunikationen mellan PLC:n och robotarna så smidig som möjligt kommer används programnummer och parametrar. Programnummren säger till robotarna vilken instruktion de skall utföra och parametrarna berättar på vilken plats något skall hämtas eller lämnas. För att få struktur på koden och för att enkelt kunna lägga till nya funktioner för en robot så är koden upplagd likadant i alla robotarna. Om en ny funktion skall läggas till så skapas en ny process i modulen som har hand om dessa och lägger till ett programnummer kopplat till denna. Exempel på kodinstruktioner och `Procedure` finns i appendix A respektive B

5.2.1 Programnummer

För att se till att robotarna utför rätt `Procedure` skickas ett programnummer från PLC:n. Robotarna tar emot programnummret och skriver över det på en intern variabel. Variabeln används sedan i en `TEST-CASE` sats för att avgöra vilken `Procedure` som skall köras. Ett exempel på hur programnummren kollas finns i appendix B.

5.2.2 Direktioner

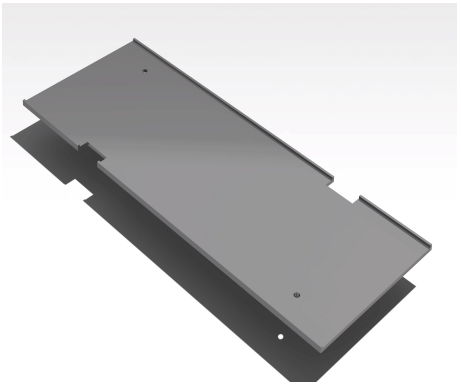
Ett problem uppstår i och med att R4 och R5 har så många möjliga rörelsekombinationer. R4 har 32 hämtningsplatser och 16 lämningsplatser, vilket leder till 512 möjliga rörelser. För att lösa problemet använder R4 och R5 en `Procedure` som tar i ett extra värde från PLC:n som säger var roboten skall hämta/ lämna en kloss. Dessa parametrar används i en `TEST-CASE` sats och sätter vart roboten skall åka. Med den här metoden behöver man bara en `Procedure` för att hämta en kloss istället för 32 olika. Om roboten får in en felaktig parameter sätts alla `Robtargets` till en hemposition och roboten ger en varning om att fel parameter har kommit in.

5.3 Verktyg & Fixturer

För att transportera klossarna och tornen in och ut ur i cellen används paletter och byggplattor. De konstruerades på ett sådant sätt så att de lätt kan förflyttas och fixerat vid sina respektive arbetspunkter. Robotarna har verktyg för att lyfta klossar, paletter och byggplattor. Byggplattor, fixturer, robotverktyg, paletter och verktygsställ behövs för att robotorna ska kunna testköra. Materialet på byggplatta och paletter är stål, dels på grund av sin styvhet men också för dess magnetiska egenskaper som hjälper klossarna att ligga stilla.

Byggplattan

Byggplattan är en 500x200 mm stålplåt som placeras på plats E, se figur 1 i kapitel 2. Plattans kanter är bockade för att ge styvhet och reducera vibrationer som kan förekomma när roboten lyfter plattan. Det är på denna platta som tornen byggs på. När tornet är klart flyttas plattan ut till operatören och en ny platta skickas in. Bilder på byggplattan ses i figur 16.



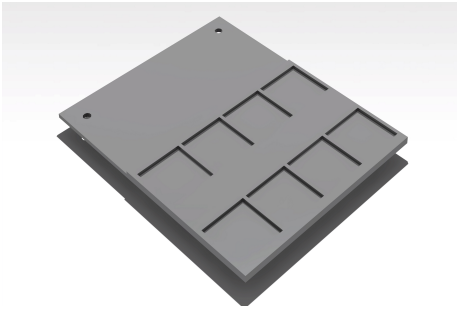
(a) Byggplatta-CAD

(b) Byggplatta

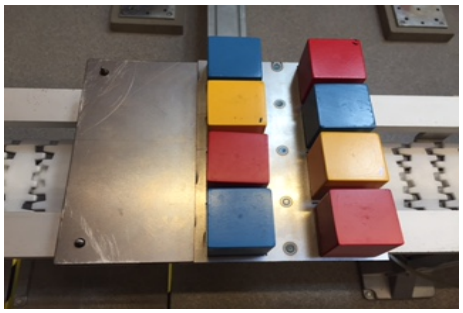
Figur 16: En CAD-modell av byggplattan och byggplattan fixerad vid byggplatsen

Paletter

Fixturerna är 220x180 mm stålplattor som transporterar klossarna runt i cellen. Dessa placeras på platserrna A-D, se figur 1 i kapitel 2. Ovanpå finns det ett 2x4 rutnät, dels för att underlätta operatörens lastning, men också för att se till att klossarna ligger still under transporten in i cellen. När operatören har lastat klart en palett så åker den på FlexLink-banan till en hiss, där en robot plockar in paletten i cellen och placerar den på rätt ställe. Bilder på paletterna finns i figur 17.



(a) CAD-modell av paletten

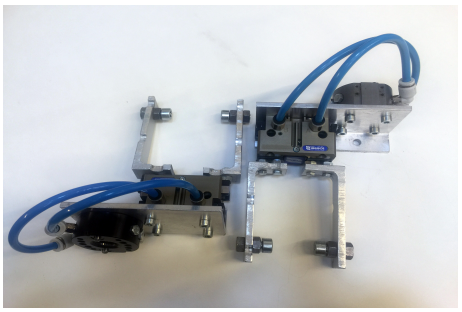


(b) Palett

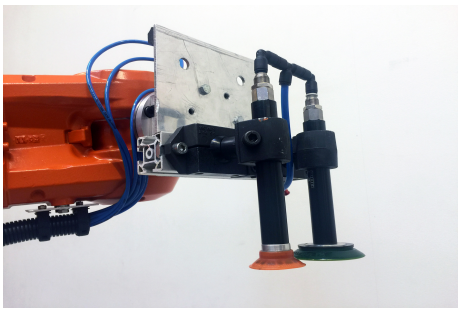
Figur 17: Modell av palett och en palett på FlexLink-bandet

Verktyg

Sammanlagt används 3 olika verktyg, ett till varje robot. R2 använder en sugkopp för att plocka upp paletterna och byggplattorna. R4 och R5 använder varsin gripklo för att plocka och placera klossarna. Klorna har modifierats då avståndet mellan armarna var för stort för att kunna gripa tag i klossarna.



(a) Gripklor



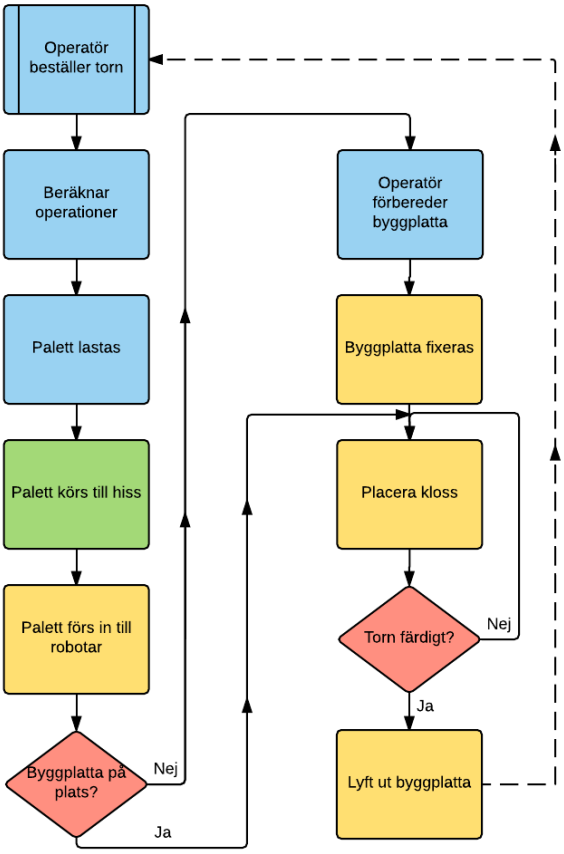
(b) Verktyg för Robot R2

Figur 18: Verktyg för R2, R4 och R5

6 Diskussion & Slutsats

Sammanställning

Styrningen av hela systemet delas upp i en tredelad hierarki: *Högnivåstyrning*, *PLC-styrning* och *robotstyrning*. I toppen av hierarkin finns den övergripande styrningen, högnivån. På denna nivå tas data in från operatören vid beställningar av torn för att schemalägga vilka operationer som behöver göras. Högnivåstyrningen beslutar när operationer ska utföras, i alla nivåer i hierarkin. Under högnivåstyrningen ligger styrningen som utförs av PLC:n. Förutom att ge anrop till robotarna så styr den FlexLink-bandet, hissarna och stoppen längs bandet. PLC:n läser in data till systemet via olika givare runt om i cellen. Längst ner i hierarkin finns robotarna som har i uppgift att bygga tornen. Robotarnas kod styr de banor som behövs för att hämta och lämna de klossar och plattor som används vid tillverkning. Dessa processer startas av PLC:n. Vissa operationer som var tänkte att startas av högnivån har istället implementerats i PLC:n och startar direkt då dess precondition är uppfyllt. Figur 19 visar flödet för en produkt i tillverkningen.



Figur 19: Schematisk bild av produktionsflödet

6.1 Diskussion

I PLC:n har operationer implementerats med generella, återanvändbara funktionsblock. Denna metod är något tidskrävande för att definiera nya resurser och operationer. Det blir däremot enklare att implementera flera instanser av samma resurs eller operation.

Strukturen med återanvändbara funktionsblock och en systematisk arkitektur i alla operationer ger en tydlig överblick och en återanvändbarhet eftersom det bara är att skapa en ny instans om en ny operation eller resurs ska implementeras. Operationerna är inte sekvensiellt ordnande utan kan startas av högnivån om systemet är i rätt state, vilket gör systemet flexibelt.

Ett resultat av ett system som kan schemaläggas är ett flexibelt system som kan anpassa sig till de förutsättningar som finns för tillfället. Detta medför krav på schemaläggningen: systemet behöver modelleras och produktionsflödet behöver simuleras och analyseras.

Sökalgoritmen som ej blev färdig var tänkt att få en generell implementation, se kapitel 3.3.3. Då hade den kunnat användas för att söka igenom liknande simuleringsmodeller, av den typ som är beskriven i schemaläggningen, vilket hade gjort den koden återanvändbar i andra projekt.

Simuleringsmodellen hade varit för specifik för att kunna återanvändas i nya projekt. Dock var det den del som tog mest tid och den blev för övergripande. Det hade varit mer effektivt att sammanfoga några operationer till en operation för att få en enklare modell. Ett sådant exempel hade varit att skicka in klosspaletter till byggplatsen. Den nya operationen hade innefattat vilken palett som operatören skickar in, att hissen skulle hissats när det finns något att hissa upp och slutligen att den stora roboten hade flyttat klosspaletten till den önskade positionen. Det är inte alltid att det är fördelaktigt att sammanfoga operationer då möjliga scheman kan försvinna men det gör modellen mer enkel, överskådlig och mindre tidskrävande att implementera.

För att robotarna inte ska köra in i samma område samtidigt så är ett kösystem hårdkodat i PLC:n. Kösystemet hade kunnat optimeras om det implementerats på högnivån istället, som då kunnat ha med detta i schemaläggningen.

En möjlig förbättring av PLC-programmering skulle vara att implementera automatiskt genererad kod för att minska tiden det tar att programera repetitiv kod. Återanvändbara och generella funktionsblock är en förutsättning för att detta skall gå.

Användargränssnitt för att beställa produkter och för att presentera instruktioner för operatören har tagits fram. Det finns även ett användargränssnitt där operatören kan se status på vissa signaler i cellen och starta enskilda operationer. Dessa tre användargränssnitt fyller sin funktion men det har inte gjorts någon metodisk

utvärdering av gränssnittens användarvänlighet. Frågor som togs upp i problemställningen var bland annat hur man gör ett gränssnitt tydligt för en operatör. För att testa detta borde någon form av vetenskaplig undersökning göras för att utvärdera användarvänligheten. Förutom tydligheten i gränssnittet så skulle det kunna diskuteras om det ska vara ett gränssnitt för varje funktion, som det är idag, eller om en eller flera av dem ska kombineras i ett gemensamt gränssnitt.

De verktyg som används av robotarna för att flytta klossar är gripklor som har anpassats för att passa måtten på klossarna. På grund av gripklornas utformning så begränsas byggandet av att tornen måste byggas rad för rad. Ett alternativ skulle vara att ha någon form av sugkopp som kan lyfta klossarna på toppen och är tillräckligt lång för att inte välta klossar som tidigare blivit placerade. Detta hade gett större flexibilitet i schemaläggningen men även fler olika sätt att bygga ett torn på vilket skapar fler möjliga scheman.

På grund av att systemet är uppbyggt med operationer så kan robotarna, efter instruktion av PLC:n, plocka en godtycklig kloss på de inmatade paletterna. Detta gör att högnivån kan bestämma helt var klossarna ska ligga och i vilken ordning de ska plockas.

Flexibiliteten i systemet gör att högnivån kan bestämma om det ska vara en eller två robotar som bygger det aktuella tornet. Om det går snabbare att använda två robotar istället för en har ännu inte testats. Det är implementerat så att den ena roboten kan hämta en kloss samtidigt som den andra sätter en kloss på tornet. Denna parallellkörning kan leda till en snabbare byggprocess.

6.2 Slutsats

Att modellera ett produktionssystem med hjälp av operationer gör de möjligt att producera produkter där ordningen på operationerna i systemet varierar. Detta sätter dock stora krav på schemaläggningen och den överordnade styrningen för att hantera varje enskilt produktalternativ.

Referenser

[1] M. G. Mehrabi, A. G. Ulsoy, och Y. Koren, ”Reconfigurable manufacturing systems and their enabling technologies”, International Journal of Manufacturing Technology and Management, vol. 1, nr 1, s. 114–131, jan. 2000.

[2] ”Industry 4.0: The Further Development of Lean Factory and Kaizen Manufacturing”. [Online]. Tillgänglig vid: http://www.azom.com/article.aspx?ArticleID=12224. [Åtkomstdatum: 15-apr-2016].

[3] M. Alpman, ”Här är Tysklands Industri 4.0”, Ny Teknik. [Online]. Tillgänglig vid: http://www.nyteknik.se/automation/har-ar-tysklands-industri-4-0-6397956. [Åtkomstdatum: 14-maj-2016].

[4] MacDougall, W. (Juli 2014) Germany Trade and Invest, INDUSTRIE 4.0 SMART MANUFACTURING FOR THE FUTURE.

[5] P. Kostal och K. Velisek, ”Flexible manufacturing system”, World Academy of Science, Engineering and Technology, vol. 53, s. 825–829, 2010.

[6] G. Langer, ”Intelligence in Manufacturing: What are Holonic Manufacturing Systems?”, Intelligence in Manufacturing, 19-feb-2008. [Online]. Tillgänglig vid: http://manufacturingintelligence.blogspot.com/2008/02/what-is-holonic-manufacturing-systems.html. [Åtkomstdatum: 12-maj-2016].

[7] K. Bengtsson, Flexible design of operation behavior using modeling and visualization. Göteborg: Chalmers University of Technology, 2012.

[8] ”What is Scala? | The Scala Programming Language”. [Online]. Tillgänglig vid: http://www.scala-lang.org/what-is-scala.html. [Åtkomstdatum: 15-apr-2016].

[9] K. Erciyes, ”Graphs”, i Distributed Graph Algorithms for Computer Networks, London: Springer London, 2013, s. 11–21.

[10] K. Erciyes, ”Routing”, i Distributed Graph Algorithms for Computer Networks, London: Springer London, 2013, s. 83–96.

[11] E. R. Alphonsus och M. O. Abdullah, ”A review on the applications of programmable logic controllers (PLCs)”, Renewable and Sustainable Energy Reviews, vol. 60, s. 1185–1205, juli 2016.

[12] W. Bolton, ”Chapter 1 - Programmable Logic Controllers”, i Programmable Logic Controllers (Sixth Edition), Boston: Newnes, 2015, s. 1–22.

[13] W. Bolton, ”Chapter 5 - Ladder and Functional Block Programming”, i Programmable Logic Controllers (Sixth Edition), Boston: Newnes, 2015, s. 115–150.

[14] W. Bolton, ”Chapter 6 – IL, SFC, and ST Programming Methods”, i Programmable Logic Controllers (Sixth Edition), Boston: Newnes, 2015, s. 151–185.

[15] ”Industri 4.0, vad krävs för att det inte ska bli kejsarens nya kläder?” [Online]. Tillgänglig vid: http://www.prevas.se/industri_4_0.html[Åtkomstdatum: 16-maj-2016].

[16] M. Anulf, P. Johansson, S. Kardell, och E. Norén, ”Flexibel utveckling av automatiserade produktionssystem: Modulärt styrkoncept och automatiskt genererad kod”, Chalmers tekniska högskola, Institutionen för Signaler och System; http://studentarbeten.chalmers.se/publication/

219268-flexibel-utveckling-av-automatiserade-produktionssystem. [Hämtad: 15 apr 2016].

[17] ABB Robotics, Technical reference manual: RAPID Instructions, Functions and Data types, uppl. J , 2010 [E-bok] Tillgänglig: https://library.e.abb.com/public/688894b98123f87bc1257cc50044e809/Technical%20reference%20manual_RAPID_3HAC16581-1_revJ_en.pdf.

Bilaga A Robotbilaga

I detta appendix finns det ytterligare information om robotarna i cellen, samt om de datatyper och instruktioner som använts i RAPID-programeringen.

Tabell 1: Robotarnas tillverkare och modellnummer

R1	KUKA	KR 30-3
R2	ABB	IRB4600 40/2.55
R3	ABB	IRB1600 8/1.45
R4	ABB	IRB140
R5	ABB	IRB140

Tabell 2: Urval av datatyper i RAPID

Instruktion	Förklaring
`TCP`	`TCP` står för **T**ool **C**enter **P**oint och definierar var ett verktygs ärbetspunktbefinner sig relativt roboten. En arbetspunkt kan vara spetsen på en svets eller undersidan av en sugkopp.
`robtarget`	En `robtarget` definierar en punkt i rummet. Vi denna punkt skall TCP och axeldata hor roboten sammanfalla med de som finns i `robtarget`
`tooldata`	En `tooldata` anänds för att definiera ett verktyg. Varje `tooldata` innehåller en `TCP` samt ett eget koordinatsystem. Desstuom innehåller en `tooldata` information om verktygets vikt och tyngdpunkt.
`speeddata`	En `speeddata` anger vilka hastigher roboten skall använda för att flytta `TCP`, omorientera verktyget samt för att vrida och linjärförflytta externa axlar.
`zonedata`	En `zonedata` anger hur nära en punkt en robot kan börja titta på nästa operation. Detta innebär att roboten borjar på nästa rörelse innan den aktuella är färdig, vilket leder till mjukare banor.

Tabell 3: En överblick av några vanliga RAPID-instruktioner

Instruktion	Exempel	Förklaring
`MoveC`	`MoveC R2BetweenTableAndBP_p20, R2BetweenTableAndBP_p10, v300, z50, sugproppgreen;`	`MoveC` intruktionen utför en cirkulär rörelse där `TCP` förblir oförändrad relativt cirklen. `MoveC` instruktionen kräver en slutpunkt och en överflygningspunkt.
`MoveJ`	`MoveJ R4HomePos, v300, z50, tool0;`	`MoveJ` instruktionen utför en icke-linjär rörelse mellan två punkter. `MoveJ` tar den bana som för roboten tar kortas tid, samtidigt som den ser till att alla aktiva leder stannar samtidigt.
`MoveL`	`MoveL R2HomeAboveBP, v200, z50, sugproppgreen;`	`MoveL` instruktionen för `TCP` längs en linjär bana mellan två punkter. Om `TCP` är stationärt kan `MoveL` användas för att omorientera verktyget kring en eller flera axlar.
`SetDO`	`SetDO DO_R4_GripOn, 1;`	`SetDO` används för att ändra värdet på en boolsk utsignal. `SetDO` kan också kombineras med tidsfördröjningar.
`SetGO`	`SetGO R4GUT_PrgNr,Order;`	`SetGO` fungerar på samma sätt som `SetDO`, men flera bitar sätts på samma gång. `SetGO` kan användas för att skicka större tal, beroende på storleken på utsignalsgruppens storlek.
`TEST`	`TEST FaParameter()`	`TEST` instruktionen används när nästa instruktion beror på en viss indata eller parametervärde. En `TEST`-sats jämför ett värde eller uttryck med ett antal unika `CASE`-värden. Om någon av `CASE`-värdena uverensstämmer utförs instruktionerna som hör till `CASE`-satsen. Om ingen matchning hittas utförs `DEFAULT`.
`WaitTime`	`WaitTime timeToWait;`	Programmet väntar vid en `WaitTime` instruktion i en viss tid. Man kan även använda en `WaitDI` för att stanna tills en viss signal fås.

Bilaga B RAPID-kod

```
MODULE GOR_UPPDRAGR2
    PROC GorUppdrag(num Uppdrag)
        !*********************************************************
        !Funktion: Väljer uppdrag efter uppdragsnummer från PLC
        !*********************************************************
        !
        ! Sätter att roboten lämnar hemmaläge
        !
        !
        ! Väljer uppdrag efter uppdragsnummer från PLC
        TEST Uppdrag

        CASE 21:
            FromBPToTable;
            PickUpFilledPalett;
            FromTableToBP;
        CASE 22:
            FromBPToTable;
            PutAwayEmptyPalett;
            FromTableToBP;
        CASE 23:
            PlacePlateAtBP(FaParameter());
        CASE 24:
            GetPlateAtBP(FaParameter());
        CASE 25:
            FromBPToTable;
            PlaceBPAtGate;
            FromTableToBP;
        CASE 26:
            FromBPToTable;
            GetBPAtGate;
            FromTableToBP;
        CASE 27:
            FromBPToTable;
            SwitchBP;
            FromTableToBP;
        CASE 28:
            FromBPToTable;
        CASE 29:
            FromTableToBP;
        CASE 30:
            INIT;
        DEFAULT:
            TPWrite "Ogiltigt programnummer. Programnumret skall
            vara ett nummer mellan 21 och 30.";
            TPWrite "Programnumret var istället " \Num:=Uppdrag;
        ENDTEST
    RETURN;
    ENDPROC
ENDMODULE
```