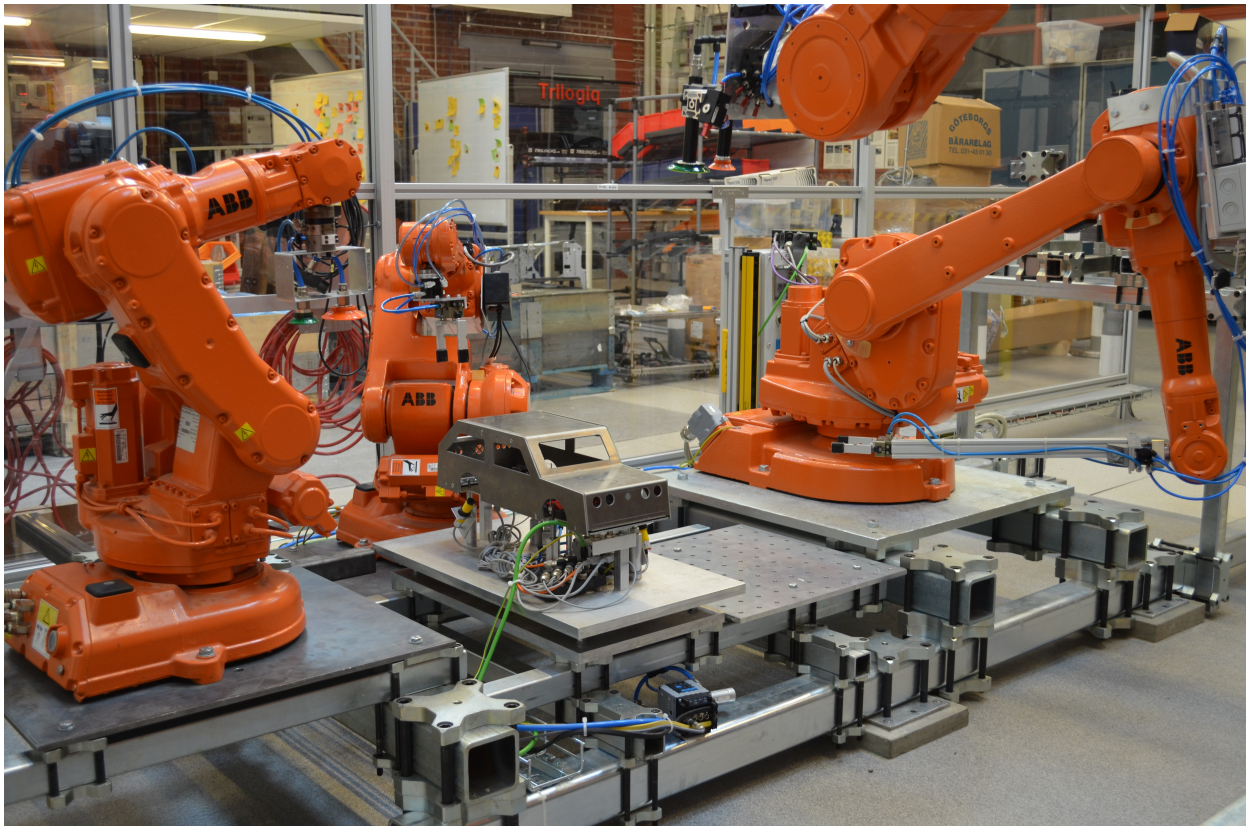
CHALMERS



Utveckling och implementering av metoder för enkel styrning och återstart av automatiserade produktionssystem

Kandidatarbete inom automation

DANIEL NORD
NIKLAS BORG
NIKLAS SKOG LIDANDER
WOLGAN DOWLAND HERRERA

Institutionen för Signaler och system
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2014

Förord

Denna rapport är del av ett kandidatarbete som utfördes våren 2014 på Chalmers Tekniska Högskola. Åtskilliga personer har hjälpt gruppen under projekets gång.

Vi vill tacka följande personer

Kristofer Bengtsson, för handledning av projektet och hjälp med PLC:n

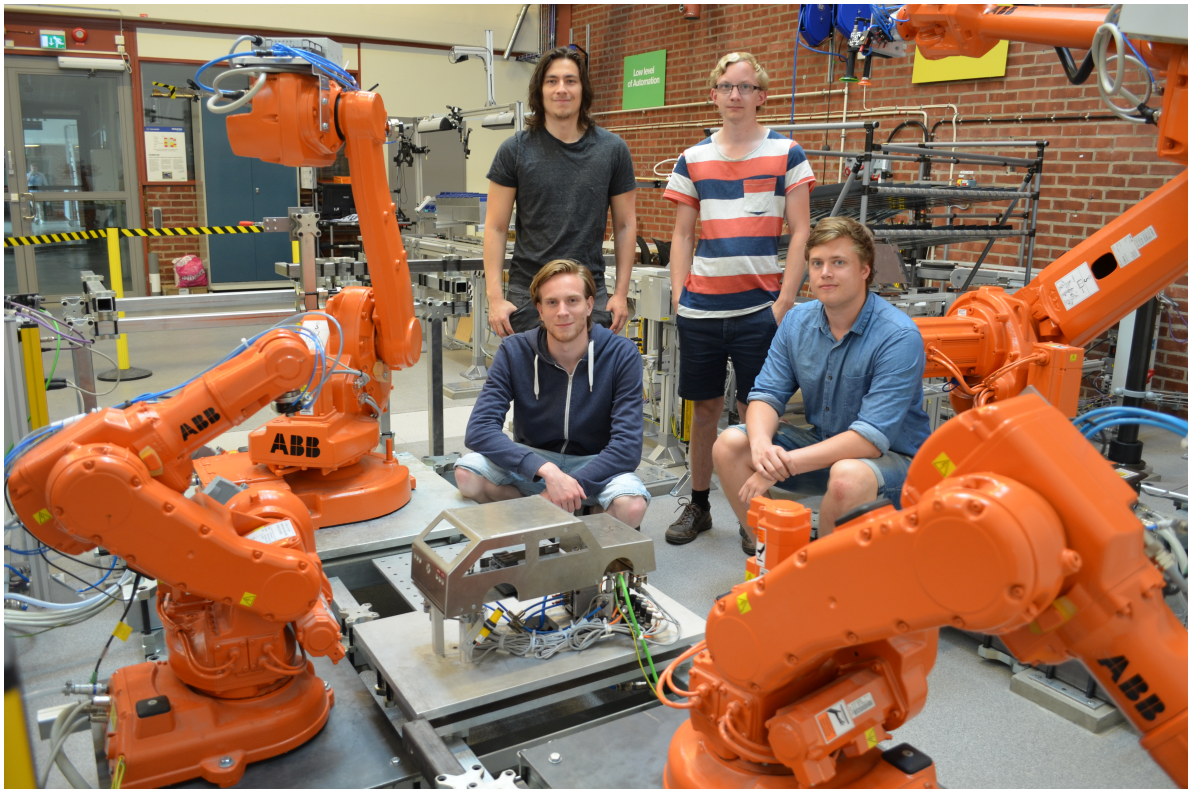
Per Nyqvist, för hjälp med och utbildning i robotik

Hans Sjöberg, för hjälp med materialinköp och diverse upplysningar om PSL

Reine och *Jan* i Prototyplabbet, för hjälp vid tillverkningen av stativen

Claes Ohlsson, för synpunkter på rapporten

Patrik Bergagård, för introduktion i sin forskning om återstart



Abstract

Complex restarts after stops in automated manufacturing systems are costly for the manufacturing industry. This thesis aims to ease restarts of such systems. To reach the aim of the project, a fit-for-purpose code structure and an innovative Graphical User Interface (GUI) are needed.

The resources of the production system (robots, conveyors etc.) are modelled with needed state variables and then programmed with "abilities". A robot can e.g. have the ability to reposition itself. One or more "abilities" are then used by one or more so-called "operations". An operation consists of a "pre-" and a "postcondition" inbetween the manufacturing of a product is advanced. For example, an operation could be to mount a car roof, but refuse to start if there aren't any floor in place to mount the roof upon.

To detect errors and ease restarts, the resource's state variables are split into a physical and a virtual model. Further, the operations are programmed to update the virtual model in the same way as the physical model was expected to be changed. Doing this enables a comparison of the models. If the models goes out of sync, a by alarm is triggered. Restarts are then eased by visualizing of the models in the GUI. The operator could then resync the physical and virtual models by manually executing the operations necessary.

The developed functionality have been implemented and tested in a manufacturing cell in the Production Systems Laboratory (PSL) at Chalmers University of Technology. Calculation of restart states and evaluation under industrial conditions are outside the scope of this thesis.

Sammandrag

Komplexa återstarter efter stopp i automatiserade produktionssystem är ett kostsamt problem för industrin. Denna rapport fokuserar på att förenkla återstart av sådana system. För det krävs en ändamålsenlig kodstruktur tillsammans med ett tillhörande, innovativt användargränssnitt.

Produktionssystemets "resurser" (robotar, transportband etc.) modelleras med nödvändig mängd tillståndsvariabler och programmeras med olika så kallade "förmågor". En "förmåga" för en robot kan exempelvis vara en bestämd förflyttning eller öppning/stängning av robotens gripverktyg. En eller flera "förmågor" utnyttjas därpå av en eller flera så kallade "operationer". En operation består av ett "start-" och ett "slutvillkor" mellan vilka tillverkningen av en produkt förs framåt. En operation kan exempelvis montera ett biltak men vägra att starta om ett golv ännu inte monterats.

För att detektera fel och underlätta återstart delas resursernas tillståndsvariabler in i en fysisk och en virtuell modell. Vidare programmeras operationerna för att uppdatera den virtuella modellen på det vis som de verkliga sensorerna förväntas ändras. Genom detta blir jämförelse av ett verkligt och ett förväntat beteende möjligt. Ett larm utlöses vid avvikelser mellan modellerna. Återstart underlättas därpå genom att de båda modellerna visualiseras i användargränssnittet. Operatören kan då återföra produktionscellen till modellens tillstånd genom manuell exekvering av lämpliga operationer.

Den framarbetade modellen har implementerats och testats i en tillverkningscell för montering av en miniatyrbil vid Production Systems Laboratory, Chalmers. Rapporten innefattar inte automatisk beräkning av lämpliga återstartslägen eller utvärdering i verklig produktion.

Innehåll

Beteckningslista

Nedan presenteras de förkortningar som används i rapporten.

Tabell 1: Använda förkortningar

Förkortning	Förklaring
ABB	Svensk verkstadskoncern som bl.a. tillverkar industrirobotar.
CAD	Computer-aided design, datorstödd ritning.
CATIA	Computer Aided Threedimensional Interactive Application, ett CAD-program.
FSA	Finite State Automaton, automaton med begränsat antal tillstånd.
GUI	Graphical User Interface, grafiskt användargränssnitt.
KUKA	Tysk industrirobottillverkare
OLE	Object Linking and Embedding
OPC	OLE for Process Controll
PDA	Pushdown Automation
PLC	Programmable Logic Controller
PSL	Production Systems Laboratory, lokal/resurs vid Chalmers Tekniska Högskola
RFID	Radio Frequency Identification
SCT	System Control Theory, reglerteknisk teori
SOP	Sequences of Operations
TIA	Totally Integrated Automation, programvara från Siemens

1 Inledning

1.1 Bakgrund

Industriautomation för med sig många fördelar. Ofta gynnas både effektivitet och kvalitet (Groover 2007). Samtidigt kan många ogynnsamma arbetsförhållanden undvikas, exempelvis ohälsosamma miljöer, tunga lyft och monotona rörelser som inte lämpar sig för människor (Lee 2012). I västvärlden har ökad industriautomation av dessa skäl kommit att bli ett viktigt vapen mot flytt av produktion till låglöneländer (Levitt 1983).

Samtidigt är tekniken för industriautomation relativt ung och har fortsatt stor utvecklingspotential. Problem finns bland annat med att enskilda robotar eller hela celler inte nyttjas fullt ut (Davidor 1991), förbrukar onödigt mycket energi (Beck och Gohner 2010) och är tidsödande att programmera.

En annan utmaning inom industriautomation rör återstart av tillverkning efter att ett fel har uppstått. Ett fel kan till exempel bestå i att en robot tappar en produkt eller att ett verktyg går sönder. Ofta dröjer det innan en operatör upptäcker ett sådant fel och stoppar produktionen. Felet kan då hunnit fortplanta sig genom systemet, vilket gör den kommande återstartsfasen svårhanterlig. Ofta tvingas operatörer till nollställning av hela linjer för att produktionen ska kunna återupptas (Andersson, Lennartsson och Fabian 2010).

1.2 Syfte

Projektet syftar till att utveckla operatörsstöd som underlättar styrning och återstart av automatiserade produktionssystem.

1.3 Mål

Syftet ska realiseras genom utveckling av en ändamålsenlig kodstruktur, ett lättanvänt användargränssnitt och en effektiv återstartsfunktion.

Kodstrukturen ska

- vara flexibel för ändring av produktionsupplägget
- snabbt detektera fel i tillverkningen
- underlätta återstart

Användargränssnittet ska

- vara intuitivt
- enkelt kunna anpassas och förändras
- erbjuda snabb åtkomst till all funktionalitet

Den utvecklade funktionaliteten ska implementeras och utvärderas i en tillverkningscell vid Production Systems Laboratory, Chalmers.

1.4 Avgränsningar

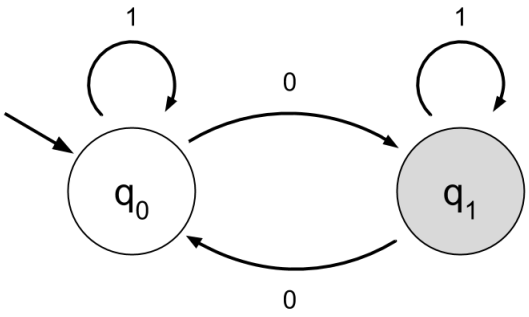
Denna rapport behandlar inte automatisk beräkning av lämpliga återstartslägen. Arbete som kräver System Control Theory (SCT) ligger utanför projektets ramar. Ej heller kommer resultaten utvärderas i industrin eller av verkliga operatörer.

2 Litteraturstudie

I detta kapitel presenteras den teori som ligger till grund för arbetet.

2.1 Automata

Finita automater är matematiska modeller som beskriver tillstånd och förändringar genom diskreta steg. Den används för att beskriva ett systems logiska beteende i dess övergångar mellan olika tillstånd. Det finns flera klasser av automater anpassade för olika typer av system. Exempelvis är en Pushdown Automaton (PDA) lämplig för representation av en datastack, då modellen utökats med beteckningar för att läsa och manipulera en stack. Ett automatiserat produktionssystem, vilket är tillämpningen för denna rapport, modelleras i regel med en finit-tillstånds-automat (Finite State Automaton, FA). (Hopcroft, Motwani och Ullman 2000)



Figur 1: Exempel på en finit automaton med två tillstånd och fyra övergångar.

Figur 1 visar en enkel finit automat. Automaten består av två tillstånd, q_0 och q_1, vilka vart och ett representeras med en cirkel. Den vänstra pilen pekar ut q_0 som initialtillstånd. De andra pilarna representerar möjliga övergångar mellan tillstånden. Övergångarna är försedda med namn vilka motsvarar en eller flera händelse(r) (event(s)), i detta fall en händelse 0 eller 1. Då en händelse sker svarar automaten med en övergång mellan det nuvarande och ett nytt tillstånd. Detta nästa tillstånd kan ibland vara samma som det nuvarande tillståndet. Den grå nyansen på q_1:s cirkel visar att det är markerat som möjligt sluttillstånd (marked final). Processen kan med andra ord inte anses som slutförd innan maskinen befinner sig i tillstånd q_1.

En finit automaton definieras formellt som en 5-tupel $A := (Q, \Sigma, \delta, q_0, F)$ där
Q är den finita uppsättningen tillstånd,
Σ är den finita uppsättningen händelser (kallad alfabet),
δ är transitionsfunktionen $\delta : Q \times \Sigma \to Q$,
q_0 är initialtillståndet $q_0 \in Q$, och
F är uppsättningen markerade sluttillstånd $F \subseteq Q$.

2.2 Fysisk och virtuell modell

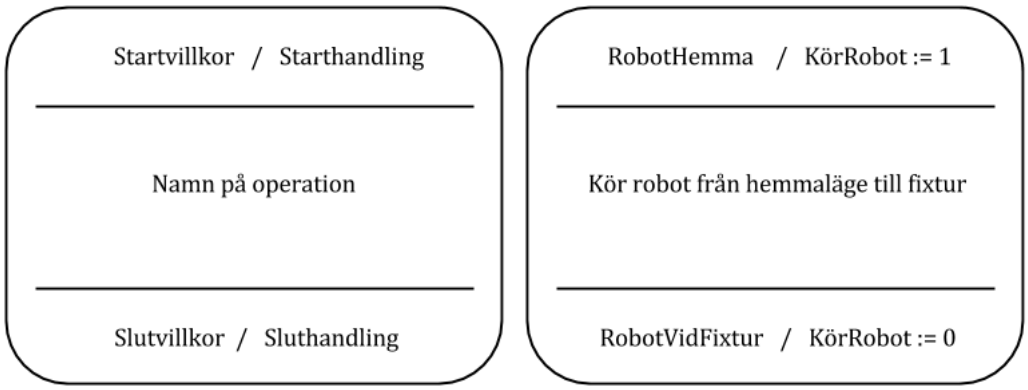
Ett produktionssystem kan betraktas som uppdelat i en fysisk anläggning (plant, P) och en kontrollant (controller, C) (Roth, Lesage och Litz 2011). Utifrån observationer av exempelvis sensorer beräknar och skickar kontrollanten styrsignaler till anläggningen. Hur det sammantagna systemet fungerar är således beroende av både P och C. Kontrollanten kan ses som det tillstånds som systemet bör befinna sig i och det fysiska är hur det ser ut egentligen (Bergagård 2013).

2.3 Operationen

Senare tids forskning, inte minst vid Chalmers, framhäver den så kallade operationen som optimal minsta byggsten vid uppbyggnad av automationssekvenser. Genom att beskriva systemets beteende med hjälp av en uppsättning operationer Ω blir systemet mer flexibelt och robust (Bengtsson 2012).

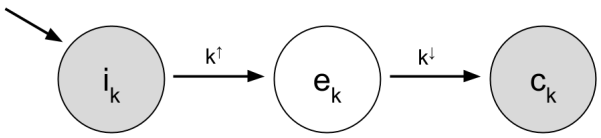
En operation $k \in \Omega$ är en aktivitet som kan utföras då dess startvillkor $k^{\uparrow}$ evalueras till sant och anses slutförd då dess slutvillkor $k^{\downarrow}$ evalueras till sant. En operation kan således befinna sig i tre tillstånd: initial- (i_k), exekverings- (e_k) och sluttillstånd (c_k). Villkoren består av boolska uttryck som bestämmer när respektive övergång kan ske. Till dessa hör också var sina serier av tillståndsförändringar som uttrycker vad som ska uträttas vid respektive övergång (Bergagård 2013). Tillståndsförändringarna benämns start- och sluthandlingar.

En operation kan till exempel bestå av att förflytta en robot från en punkt till en annan, kanske från ett hemmaläge till en fixtur. En grafisk visualisering av en sådan operation kan ses i Figur 2. I figuren är start- och slutvillkor utsatta tillsammans med start- och sluthandlingar. RobotHemma sätts till sann om roboten befinner sig i sitt hemmaläge. Detsamma gäller RobotVidFixtur som ettställs då roboten befinner sig vid fixturen. Roboten förflyttas genom att KörRobot ett- och nollställs.



Figur 2: Visualisering av en operation

En operation kan formellt modelleras som en automat kallad operationsautomat. Automaten för en operation k betecknas A_k där $Q_{A_k} := \{i_k, e_k, c_k\}$; $\Sigma_{A_k} := \{k^{\uparrow}, k^{\downarrow}\}$; $\delta_{A_k} := \{\langle i_k, k^{\uparrow}, e_k\rangle, \langle e_k, k^{\downarrow}, c_k\rangle\}$; $q^0_{A_k} := i_k$ och $Q^m_{A_k} := \{i_k, c_k\}$. Händelserna i Σ_{A_k} kallas för operationshändelser. Operationsautomatonen A_k visualiseras i Figur 3. (Bergagård 2013)



Figur 3: Visualisering av en operationsautomaton

2.4 Sekvens av operationer (SOP)

För många system blir mängden av operationer ganska stort, vilket kan göra det svårt att förstå systemets beteende. (Bengtsson 2012) Det kan då hjälpa att gruppera operationer i olika sekvenser.

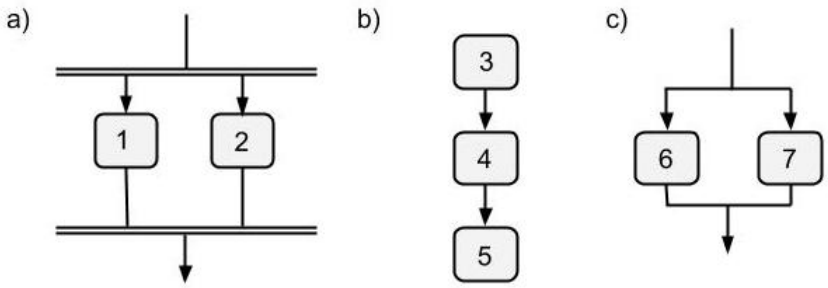
Den huvudsakliga anledningen till varför man visualiserar en grupp av operationer i en sekvens är att kunna studera deras relation till varandra (Bengtsson och Lennartsson 2013). En SOP kan göras av olika anledningar. Ett exempel på en SOP kan vara att alla ingående operationer utförs av samma robot.

Tre vanliga relationer i en SOP visas i Figur 4 nedan. Parallella, sekventiella, och alternativa relationer visas i a), b), respektive c). Dessa kan fritt kombineras till större SOP:ar. Det finns även andra relationer som inte tas upp här.

Den parallella relationen har horisontella dubbelstreck över och under operation 1 och 2. Med en parallell relation menas att operationerna innanför de horisontella strecken kan köras parallellt. Alla de ingående operationerna måste dock slutföras innan SOP:en kan fortsätta, i detta fall båda operationerna 1 och 2.

I en enkel sekventiell relation som i b) kan operationerna bara utföras i den ordning som är uppsatt. Det går till exempel inte att utföra operation 4 före operation 3 är slutförd.

I den alternativa relationen c) kan bara en av vägarna väljas. I det här exemplet körs antingen 6 eller 7. Vilken operation som väljs beror på de ingående operationernas vakter, alltså vilken som (först) evalueras till sann.



Figur 4: Visualisering relationer mellan operationer i en SOP

2.5 Feldetektering

Ett exempel på fel som kan uppstå i ett produktionssystem är att en robot tappar en plåtbit eller liknande innan den monterats. För att upptäcka sådana fel är det möjligt att separera styrningen i en fysisk modell P och en virtuell modell $\hat{P}$ (Roth, Lesage och Litz 2011).

Den fysiska modellen uppdateras kontinuerligt genom avläsning av produktionssystemets sensorer. Den virtuella modellen uppdateras av operationer. Fel kan påvisas genom skillnader mellan dessa två modeller. Skillnader mellan modellerna upptäcks genom konstant kontroll av P och $\hat{P}$ eftersom de alltid ska stämma överens (Bergagård 2013).

2.6 Återstart

En återstart av ett produktionssystem kan bli väldigt besvärlig och tidskrävande sedan ett fel fortplantat sig genom systemet. För att lösa detta krävs ofta en återställning av hela systemet, vilket i regel är mycket tidskrävande. Effektiv återstart innebär att systemet snabbt ska kunna startas om från ett lämpligt tillstånd (Bergagård, Falkman och Fabian).

Återstart av systemet kan betraktas som att en fysisk och en virtuell tillståndsmodell ska återsynkroniseras. Detta kan då göras genom antingen förändring av den fysiska verkligheten eller manipulation av den virtuella modellen. Om manipulation av den virtuella modellen tillåts kan operatören starta upp systemet från valfritt tillstånd (Bergagård 2013).

Fel kan uppstå under en operations exekveringsfas så att den aldrig når sitt sluttillstånd. Ibland kan ett sådant problem innebära att produkten måste kasseras. Detta eftersom vissa produktoperationer, så som svetsning, inte kan göras om (Bergagård 2013). Det kan också vara så att produkten inte klarar av de kravspecifikationer som satts upp om produkten korrigeras manuellt (Andersson m. fl. 2012).

2.7 Robotprogrammering

Detta projekt innefattar endast robotar av fabrikat ABB. Programmering av ABB:s robotar görs i kodspråket RAPID.

När en robot ska utföra en förflyttning används så kallade Move-instruktioner. Till dessa instruktioner tilldelas lämpliga punkter som kan ses som målkoordinater. Vilken typ av Move-instruktion som utförs avgör på vilket sätt roboten rör sig mot målpunkten. De två viktigaste utgörs av en linjärinstruktion (Linear, MoveL) och en ledinstruktion (Joint, MoveJ). Vid en linjär instruktion rör sig roboten i en linjär bana medan den vid en ledinstruktion tar den (för robotens leder) snabbaste vägen till målpunkten. När Move instruktionerna programmeras så väljs också hastighet och noggranhet till rörelsen (ABB Robotics 3-11, 3-12).

Det finns även andra instruktioner som robotarna kan utföra, bland annat SetDo-instruktioner. Dessa ett- eller nollställer robotens olika digitala utgångar. En SetDo-instruktion kan till exempel användas för att aktivera en vakuumsugpropp eller att stänga en gripklo. Några av signalerna går ut på en I/O-buss via vilken det går att skicka och ta emot signaler till PLC:n (ABB Robotics 3-30, 8-4).

Alla rörelser och SetDo-instruktioner kodas i rutiner. En rutin kan till exempel flytta en robot från punkt A till punkt B där den sedan nollställer en digital utgång C. Rutinerna placeras sedan i moduler. Det finns två typer av moduler: program- och systemmoduler. Skillnaden mellan dessa två är att systemmoduler består mellan inladdningar av olika projekt (ABB Robotics 8-4).

Det finns två huvudsakliga metoder för programmering av ABB:s industrirobotar: online- och offlineprogrammering. Onlineprogrammering innebär att en operatör står vid robotens styrenhet (FlexPendant) och manuellt kör roboten till de punkter som ska sparas. Att programmera på FlexPedanten är mycket tidskrävande (Pan 2012).

Offlineprogrammering utförs istället i en simuleringsmjukvara innehållandes en virtuell modell av produktionssystemet. Offline- och onlineprogrammering är på många sätt lika varandra. Robotarna körs till den punkt som programmeraren önskar vilket möjliggör sparande av en mål- eller viapunkt där. Skillnaden ligger bara i huruvida rörelserna utförs i verkligheten eller i simuleringsprogramvaran. I simuleringsprogramvaran kan alla rörelser testas utan risk för att förstöra roboten och produktionssystemet eller orsaka personskada. Offlineprogrammering kräver dock att den virtuella modellen stämmer väl överens med verklighetens dimensioner. Om inte uppstår

risk för både person- och maskinskador genom att rörelser som simulerats i modellen kan krocka med något i verkligheten (Ibid.).

Det är också möjligt att simulera robotens I/O-signaler tillsammans med de produkter som robotarna ska montera. Dessa I/O-signaler kopplas då till händelser som ska ske i cellen när de aktiveras, till exempel att en plåtbit ska fastna på robotens sugverktyg. Detta medför att simuleringen kan visa om roboten klarar av att till exempel montera eller hämta en produkts detaljer på avsett sätt (Ibid.).

När programmet slutligen är klart för test laddas det in i roboten via en nätverkskabel kopplad från den simulerande datorn till robotens styrskåp. Eftersom större delen av programmeringen kan utföras utan stopp i produktionen kan mycket tid och pengar sparas jämfört mot onlineprogrammering. (Ibid.).

3 Metod

3.1 Tillvägagångssätt

Arbetet inleddes med studier i aktuell forskning. Därpå implementerades utvalda ideer i ett produktionssystem vid Production Systems Laboratory (PSL), Chalmers.

3.2 Projektframdrivning

Projektet drevs fram med hjälp av Scrum, vilket är en metodik för systemutveckling. Den passar då en specifik kravspecifikation saknas till förmån för föränderliga mål. I en så kallad backlog listas lämpliga delprojekt (stories) (Axelsson 2013). Backloggen fylldes med stories allt eftersom sådana uppenbarades, exempelvis "modellera robotcellen i CATIA".

Utöver Scrum har Kanban praktiserats. Denna metod syftar till att effektivt kommunicera framsteg i arbetet både inom och utom projektgruppen, begränsa antalet stories samt tydliggöra flaskhalsar (Crisp). Kommunikationen skedde visuellt på en whiteboard-tavla utformad som Tabell 2.

Tabell 2: Kanban-uppställning med förklaringar

Backlog	Utvalt	Pågående	Klart för utvärdering	Analys	Färdigt
Här lades stories som planerades att göras och fungerade därmed som ett slags minne.	Här lades stories som var utvalda att tas itu med så fort nästa kolumn, Pågående, fick plats.	Aktiva stories placerades här. Endast ett begränsat antal stories, valfritt många, var tillåtna i denna kolumn.	Här hamnade stories som var färdigarbetade och väntande analys. Analysören är projektledaren eller handledaren.	Här hamnade stories under analys	Till sist hamnade stories som gick igenom analysen och därmed blev färdiga här.

Eftersom pågåendekolumnen endast tillät ett visst antal stories blev det tydligt om en viss story dröjde längre än andra. En sådan story gavs då mer uppmärksamhet.

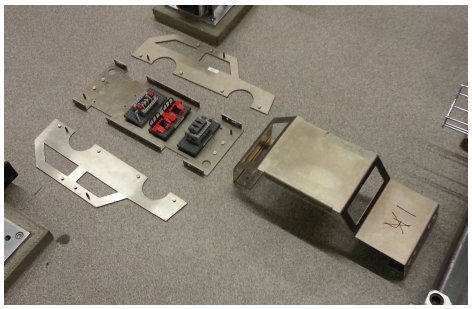
4 Produktionssystemet i PSL

Produktionssystemet i PSL, fortsättningsvis även hänvisad till som tillverkningscellen eller bara cellen, har byggts om för att bättre lämpa sig som fall för studier i effektiv återstart. Cellen monterar nu samman en miniatyrbil med hjälp av fyra robotar. I detta kapitel beskrivs utformningen av den nya cellen.

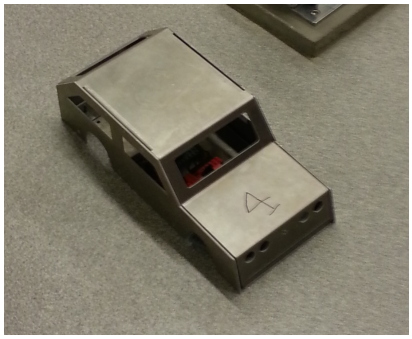
4.1 Hårdvara

4.1.1 Miniatyrbilen

Miniatyrbilen består av totalt sju delar: golv, tak, höger- och vänstersida i plåt samt växellåda, motor och sittplats byggda i lego. De olika delarna kan ses i Figur 5 och den färdigmonterade miniatyrbilen i Figur 6. Monteringen måste utföras i en viss ordning, se avsnitt 4.2. Miniatyrbilen hålls samman av små magneter och små styrpinnar som för delarna till önskad plats. Sammanfogningen kräver därför ingen särskild operation i form av exempelvis limning eller svetsning.



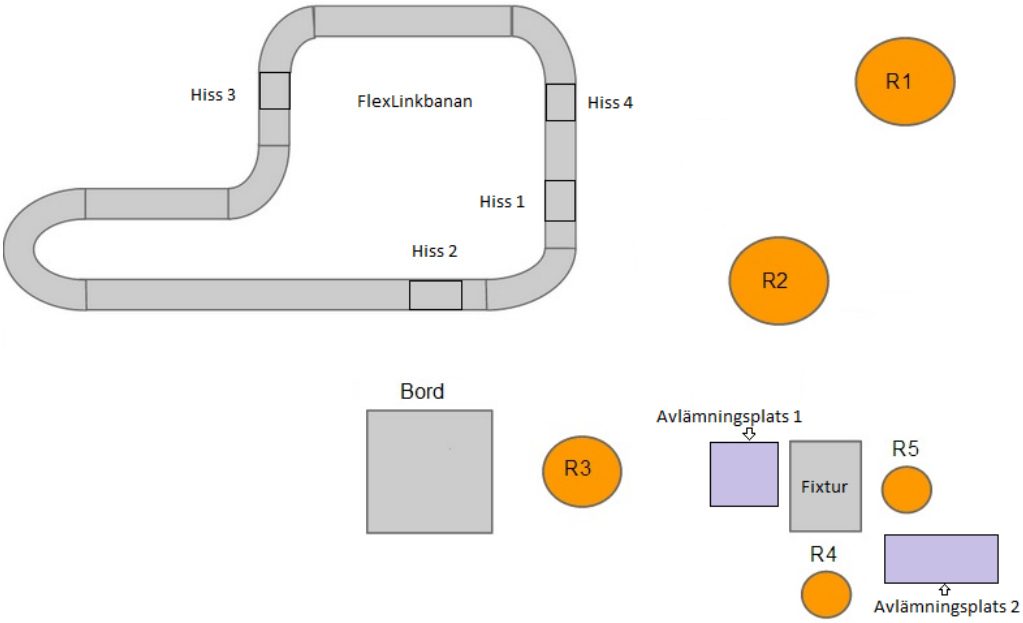
Figur 5: Miniatyrbilen isärplockad



Figur 6: Miniatyrbilen monterad

4.1.2 Produktionssystemets utformning

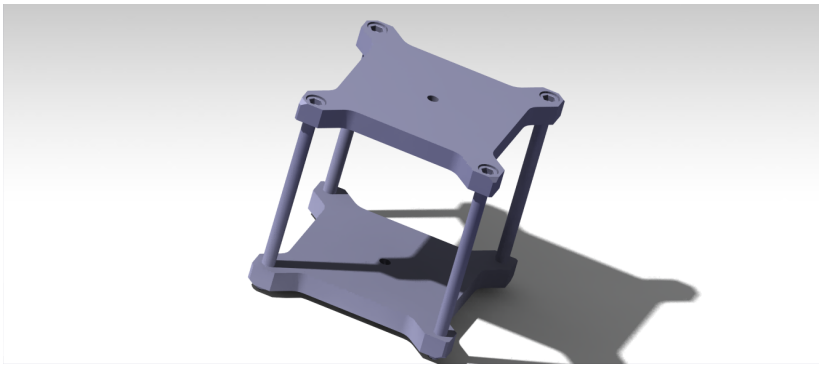
Produktionssystemet, som kan ses i Figur 7, utgör ett cirka 40 kvadratmeter stort område. Det består sammantaget av fem industrirobotar (benämnda R1-R5), en fixtur, ett bord, två avlämningsplatser och ett transportband av fabrikat FlexLink (fortsättningsvis benämnd "Flex-Linkbanan").



Figur 7: Skiss över tillverkningscellen

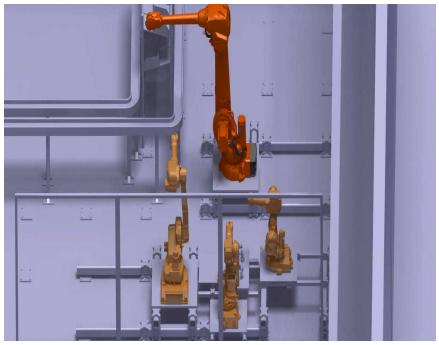
Industrirobotarna som finns i cellen är av olika fabrikat. Fyra av robotarna är levererade av ABB (R2-R5) och en av KUKA (R1). Robot R1 används dock inte i detta projekt. ABB-robotarna är av olika storlek, vilket i Figur 7 visualiseras genom olika diameter på cirklarna. Robotarna har placerats så att de alla kan nå fixturen. Att så många som fyra robotar används grundar sig i en önskan om viss komplexitet i produktionsprocessen.

Robotarna är monterade på stålbalkar som sammanfogats med så kallade "BoxJoints". Två sådana plattor med fyra skruvar kan ses i Figur 8. Fördelen med BoxJoint är att det är enkelt att justera eller bygga om konstruktionerna och att de balkar som använts kan återanvändas i andra projekt.

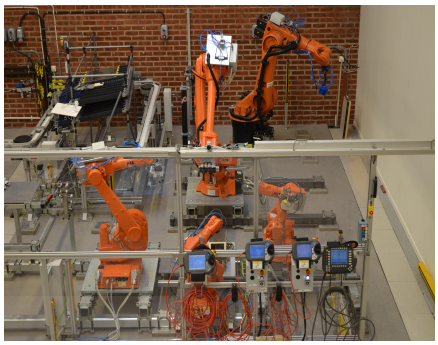


Figur 8: CATIA-modell av en BoxJoint

Den ombyggda cellen har ritats upp i CAD-programmet CATIA, vilket kan ses i figur 9. Hur det verkliga resultatet blev kan i sin tur ses i figur 10.

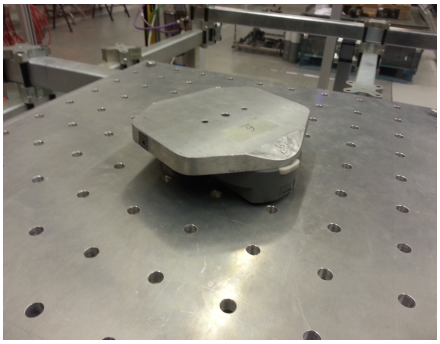


Figur 9: Modell av tillverkningscellen

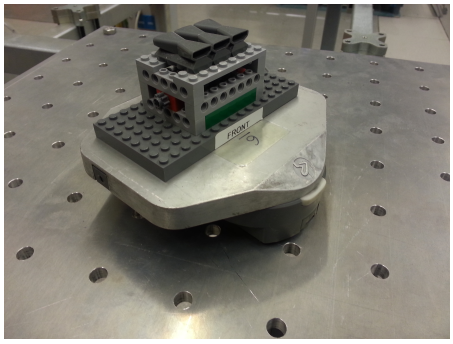


Figur 10: Det verkliga produktionssystemet

FlexLinkbanan transporterar in bildelarna till robotarna på så kallade paletter. Dessa paletter är unikt identifierbara med RFID-teknik. En tom palett kan ses i Figur 11 och en med legobit i Figur 12.

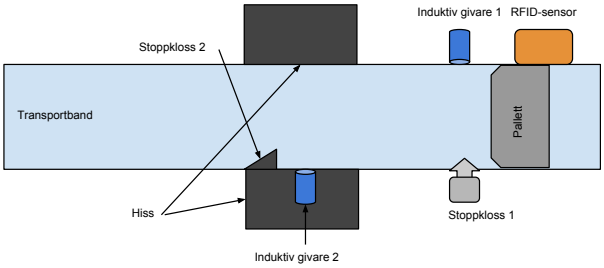


Figur 11: Palett för FlexLinkbanan



Figur 12: Palett med lego

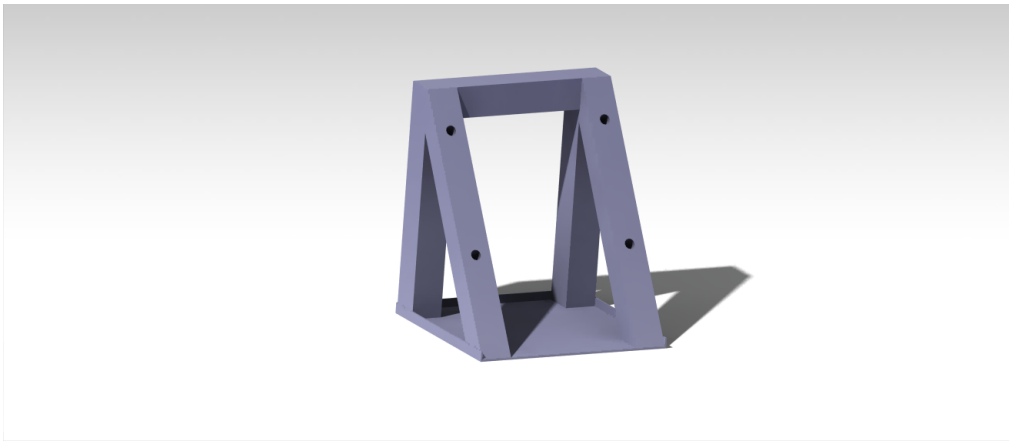
FlexLinkbanan har fyra hissar utrustade med fem komponenter vardera. De fem komponeneterna utgörs av två induktiva givare, en RFID-sensor och två stoppklossar. Stoppkloss 1 är placerad i banhöjd medan stoppkloss 2 följer hisskorgen. Placeringen av alla komponenter kan ses i figur 13.



Figur 13: Placering av komponenter vid en hiss

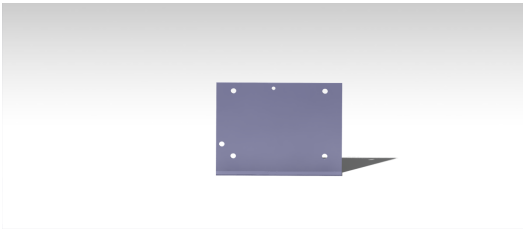
Hissarnas funktion är att lyfta upp bildelarna då robotarna ska plocka upp dessa. Stoppklossarnas funktion är att stabilisera hisslyftet respektive att hindra paletterna från att åka vidare på rullbandet.

Fyra stativ har byggts för transport av bilens karossdelar. Stativens mått har inte tillåtits skilja sig mycket åt då robot R2:s sugverktyg inte tolererar några större avvikelser. Stativen består av två delar. Den första, se figur 14, är en gemensam grunddel som fästs på paletterna. På denna kan sedan en lämplig modul snabbfästas. Snabbfästet består av magneter som håller de olika modulerna på plats.

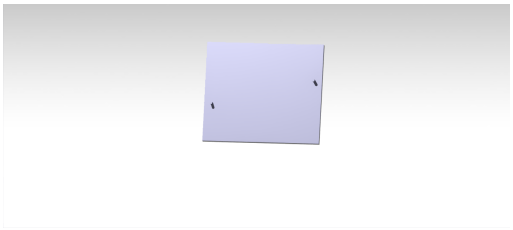


Figur 14: Modell av grundtriangel

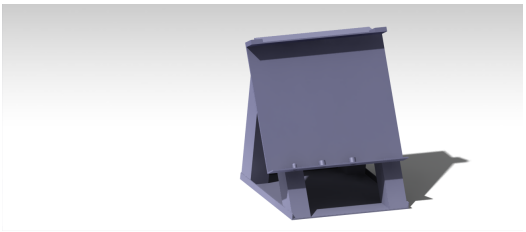
Det finns en modul för varje typ av bildel. Två av dessa moduler kan ses i figur 15 och figur 16. En grunddel med modul monterad kan ses i figur 17 och med tillhörande bildel i figur 18.



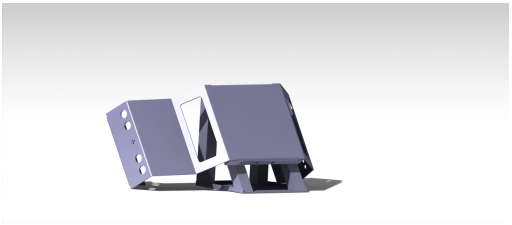
Figur 15: Modul för bilens sida



Figur 16: Modul för bilens golv



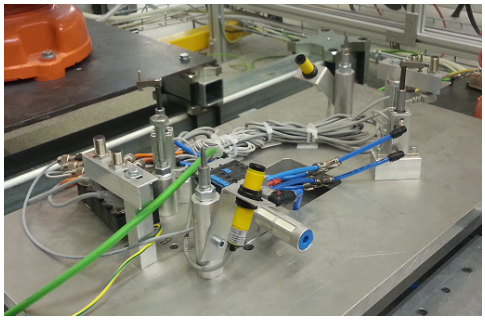
Figur 17: Takmodul utan tak monterat



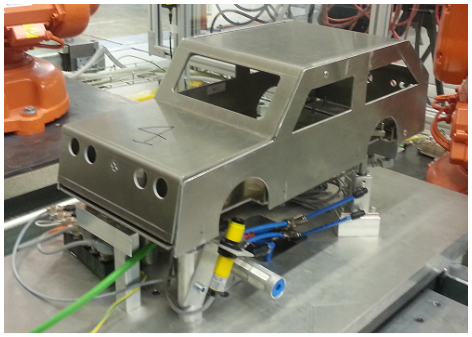
Figur 18: Takmodul med tak monterat

4.1.3 Fixturen

Monteringen av miniatyrbilen utförs i fixturen, se figur 19 och 20. Fixturen består av en platta utrustad med styrpinnar, luftkolvar och sensorer. Styrpinnarnas funktion är att styra golvet till en viss position. Luftkolvarna används därpå för att låsa fast golvet efter att det placerats i fixturen. Fastlåsningen gör att golvet blir helt fixerat när de senare delarna ska monteras. Sensorerna används för att avgöra om till exempel ett golv har monterats eller ej. Det finns minst en sensor för varje monterbar karossdel.



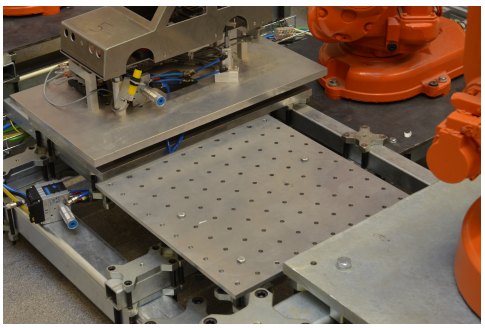
Figur 19: Fixturen utan miniatyrbil



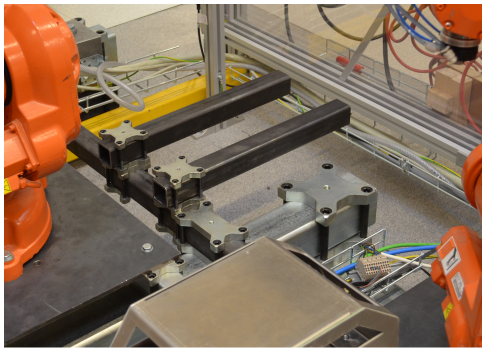
Figur 20: Fixturen med miniatyrbil

4.1.4 Avlämningsplatser

Det finns två avlämningsplatser i tillverkningscellen med var sina ändamål. Avlämningsplats 1 är en buffert för bilens sittplatser, motor och växellåda. Denna kan ses i Figur 21. Avlämningsplats 2 är avsedd för färdiga bilar vilket frigör fixturen för vidare produktion. Avlämningsplats 2 kan ses i Figur 22.



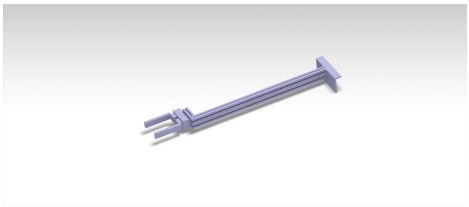
Figur 21: Avlämningsplats 1



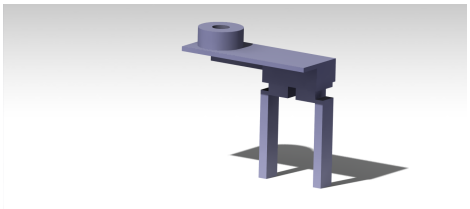
Figur 22: Avlämningsplats 2

4.1.5 Robotverktyg

Då samtliga robotar R2-R5 ska utföra liknande uppgifter har ibland befintliga verktyg antingen saknats helt eller inte kunnat utföra de nödvändiga arbetsuppgifterna. Av denna anledning har två nya robotverktyg konstruerats. Robot R3 krävde ett långt gripverktyg för att kunna nå inkommande delar på FlexLinkbanan. Detta verktyg kan ses i figur 23. Robot R4 krävde ett gripverktyg som kan plocka lego och montera den på bilen. Detta verktyg kan ses i figur 24.



Figur 23: Modell av gripverktyg för R3



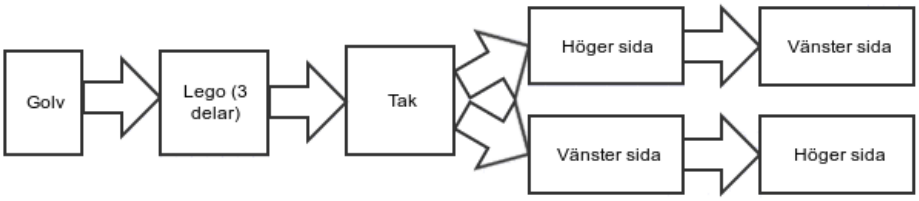
Figur 24: Modell av gripverktyg för R4

4.2 Hur miniatyrbilen monteras

Monteringen inleds genom att en operatör placerar önskad detalj på en palett och sänder iväg denna på FlexLinkbanan.

Ett krav på FlexLinkbanan är att delarna inte får krocka med varandra, då detta i verkligheten skulle motsvara en förstörd produkt. Ett annat krav är att paletterna bara ska lyftas av rätt hiss. Exempelvis ska inte legobitarna skickas upp i hiss 1 utan endast vid hiss 2. Paletterna ska vidare bara hissas upp om de för med sig en efterfrågad del. Om exempelvis hiss 1 efterfrågar ett golv ska alla andra typer av delar passera hiss 1 utan åtgärd.

Den ordning som delarna monteras i kan ses nedan i Figur 25 och beskrivs mer i nästa stycke.



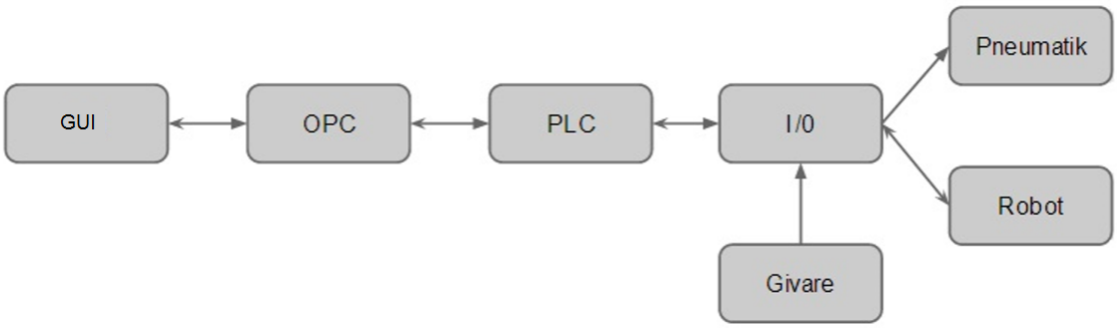
Figur 25: Ordningen som delarna monteras i

Bilens karossdelar plockas från hiss 1 och monteras i fixturen av robot R2. Legot (sittplats, motor och växellåda) plockas från hiss 2 och läggs på avlämningsplats 1 av robot R3. Legot placeras därefter på sina rätta platser på golvet av robot R4. När en bil är färdigmonterad plockar R5 upp bilen och lägger den vid avlämningsplats 2.

Under monteringen av miniatyrbilen föreligger ibland kollisionsrisk för robotarna. Därför kan robotarna inte tillåtas utföra vissa rörelser samtidigt. För att undvika kollisioner utnyttjas så kallad zonbokning. Tre zoner har skapats: en innefattande fixturen, en innefattande avlämningsplats 1 samt en för transporter utanför avlämningsplats 1. Innan en robot får utföra en rörelse kontrollerar PLC:n om zonen är ledig. Om så är fallet bokas zonen av den begärande roboten. Om zonen däremot redan är bokad tvingas roboten att vänta till dess att zonen åter är ledig.

4.3 Kommunikation i produktionssystemet

Produktionssystemet styrs av en så kallad PLC (Programmable Logic Controller) vilken är en standardiserad styrdator för industriella automationstillämpningar. Den PLC som används i PSL är levererad av Siemens, vilket också innebär att Siemens utvecklingsprogramvara Step 7 används för programmeringen. Siemens Step 7 arbetar främst med språket Ladder (LD), ett visuellt språk vars utformning är starkt influerat av gamla tiders realisering av automation genom kretslogik. PLC:n sköter all kommunikation mellan komponenterna i systemet, vilket illustreras i Figur 26.



Figur 26: Kommunikation i tillverkningscellen

4.4 Robotprogrammen

Det krävs många rutiner för att montera samman en hel miniatyrbil. I Figur 27 ses en av dessa rutiner.

```
PROC pick_carfloor()
SetDO R2UT_HomePos,0;
SetDO R2UT_FlexlinkPos,0;
SetDO R2UT_FixturePos,0;

      MoveL flexlink_wait_pos,v1000,z100,R2sug\WObj:=wobj0;
      MoveL pre_pick_floor_pos,v600,z15,R2sug\WObj:=wobj0;
      MoveL pick_floor_pos,v100,fine,R2sug\WObj:=wobj0;
            SetDO DO10_6,1;
            WaitTime 0.5;
      MoveJ pre_pick_floor_pos,v600,z30,R2sug\WObj:=wobj0;
      MoveL flexlink_wait_pos,v1000,z100,R2sug\WObj:=wobj0;

SetDO R2UT_FlexlinkPos,1;
ENDPROC
```

Figur 27: En RAPID-rutin som plockar upp ett biltak från hiss 1

Rutinen i Figur 27 använder SetDo-instruktioner för att indikera i vilket läge roboten befinner sig. Att alla lägessignaler har värdet noll ska tolkas som att roboten befinner sig i ett exekveringstillstånd.

Varje robot har, utöver ett hemmaläge, försetts med två väntlägen. Robot R2 har väntlägen vid FlexLinkbanan och fixturen, robot R3 vid FlexLinkbanan och avlämningsplats 1, robot R4 vid fixturen och avlämningplats 1 och robot R5 vid fixturen och avlämningsplats 2. Det är endast vid dessa väntlägen som roboten kan ta emot ett nytt uppdrag från PLC:n.

4.4.1 Utförande av uppdrag

I robotarna finns en programrutin som hanterar vilket uppdrag roboten ska utföra. Denna rutin väntar på ett programnummer från PLC:n vilket representerar ett uppdrag. Efter att programnumret lästs in av roboten tar rutinen reda på vilket fall som ska utföras genom att testa numret i en lång så kallad “case-sats”. Ett exempel på en sådan sats kan ses i figur 28.

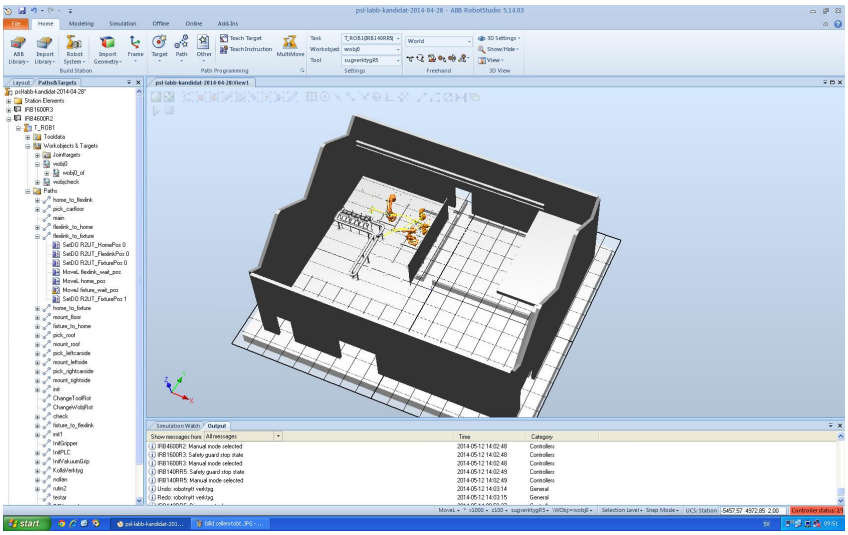
```
! Väljer uppdrag efter uppdragsnummer från PLC
TEST Uppdrag

CASE 320:
        IF DOutput(R3UT_HomePos)=1 THEN
            home_to_flexlink;
        ENDIF
CASE 321:
        IF DOutput(R3UT_HomePos)=1 THEN
            home_to_table1;
```

Figur 28: En bild på två CASE-satser som innehåller olka uppdrag.

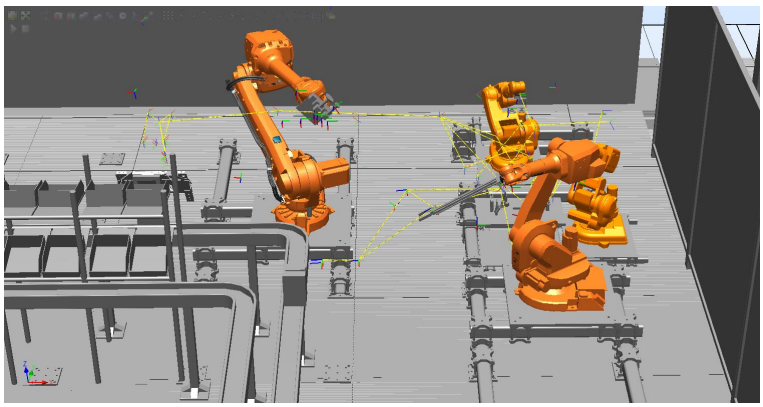
Efter att numret har lästs in kvitterar roboten läsningen genom att skicka tillbaka samma programnummer till PLC:n.

För att möjliggöra offlineprogrammering har en virtuell modell skapats i CAD-programvaran CATIA V5. Simulering har därpå utförts i ABB:s RobotStudio. Arbetsmiljön i RobotStudio kan ses i Figur 29.



Figur 29: RobotStudio med inladdad modell av PSL

Den virtuella modell som skapades i CATIA kunde ses i Figur 9. CATIA-modellen konverterades därefter med programmet "CAD Converter" för att kunna laddas in i ABB RobotStudio. Robotar placerades ut och kompletteras med verktyg för att bilda en fullständig modell. Det slutliga resultatet i ABB RobotStudio kan ses i Figur 30.



Figur 30: RobotStudio-modell av produktionssystemet

5 Resultat

I detta kapitel beskrivs utvecklade metoder för enkel styrning och återstart och hur dessa implementerats i PSL.

5.1 Indelning i fysisk och virtuell modell

Likt uppdelningen som beskrivs i teoriavsnittet 2.2 har två parallella modeller skapats för systemet: en fysisk och en virtuell.

Den fysiska modellen består enbart av variabler med koppling till fysiska sensorer och aktuatorer i produktionssystemet. Tanken är att denna modell ska avspegla produktionssystemets (verkliga) fysiska tillstånd. Kravet för att en variabel ska inkluderas i denna modell är alltså att den går att mäta i någon form.

Den virtuella modellen består tvärtom av variabler som inte sätts av eller styr något fysiskt i produktionssystemet. Denna modell speglar istället det förväntade tillståndet för produktionssystemet. Den virtuella modellen inkluderar variabler för sensorernas och aktuatorernas börvärden, men också fritt modellerade variabler vars status inte kan verifieras. I PSL motsvaras det senare exempelvis av vilken bildel roboten håller i för tillfället, vilket roboten saknar sensorer för.

I Siemens PLC-mjukvara TIA Portal kan så kallade databaser skapas. En databas består av ett valfritt antal variabler. För den fysiska och den virtuella modellen används två databaser vardera: “Controlling” och “Sensors” respektive “Sensor Duplicates” och “Modelled”.

Den förstnämnda databasen (Controlling) innehåller variabler som operatören kan sätta för att styra funktioner i produktionssystemet. En skrivning till en variabel i denna databas innebär att samma värde direkt skrivs till adressen för en fysisk funktion i produktionssystemet. I PSL har exempelvis variabler för stoppklossarna på FlexLinkbanan placerats i denna databas. En bild över databasen Controlling kan ses i figur 31.

Controlling

	Name	Data type	Offset	Start value	Retain	Visible in ...	Setpoint
1	Static				☐	☐	☐
2	F_H1_ST1	Bool	0.0	false	☑	☑	☐
3	F_H2_ST1	Bool	0.1	false	☑	☑	☐
4	F_H3_ST1	Bool	0.2	false	☑	☑	☐
5	F_H4_ST1	Bool	0.3	false	☑	☑	☐
6	F_S0_ST1	Bool	0.4	false	☑	☑	☐
7	F_S2_ST1	Bool	0.5	false	☑	☑	☐
8	F_S3_ST1	Bool	0.6	false	☑	☑	☐
9	F_F0	Bool	0.7	false	☑	☑	☐

Figur 31: Bild på databasen Controlling i TIA Portal

Värdena på variablerna i databasen Sensors sätts av fysiska sensorer i produktionssystemet. Dessa variabler kan alltså, till skillnad från de i databasen Controlling, inte ändras av operatören. En bild på databasen Sensors kan ses i Figur 32.

Sensors

	Name	Data type	Offset	Start value	Retain	Visible in ...	Setpoint	Comment
1	Static							
2	Fixture_Floor_Rear	Bool	0.0	false	☑	☑	☐	
3	Fixture_Floor_Front	Bool	0.1	false	☑	☑	☐	
4	Fixture_Roof_Rear	Bool	0.2	false	☑	☑	☐	
5	Fixture_Roof_Front	Bool	0.3	false	☑	☑	☐	
6	Fixture_Side_Left	Bool	0.4	false	☑	☑	☐	
7	Fixture_Side_Right	Bool	0.5	false	☑	☑	☐	
8	Fixture_Cylinder_Up_Front	Bool	0.6	false	☑	☑	☐	
9	Fixture_Cylinder_Down_Front	Bool	0.7	false	☑	☑	☐	
10	Fixture_Cylinder_Up_Rear	Bool	1.0	false	☑	☑	☐	
11	Fixture_Cylinder_Down_Rear	Bool	1.1	false	☑	☑	☐	
12	R2_POS	DInt	2.0	0	☑	☑	☐	0=Hemma,1=Flexlink,2=Fixtur,3=Kör
13	R3_POS	DInt	6.0	0	☑	☑	☐	0=Hemma,1=Flexlink,2=Table1,3=Kör
14	R4_POS	DInt	10.0	0	☑	☑	☐	0=Hemma,1=Table1,2=Fixtur,3=Kör
15	R5_POS	DInt	14.0	0	☑	☑	☐	0=Hemma,1=Fixtur,2=Table2,3=Kör

Figur 32: Bild på databasen Sensors i TIA Portal

Databasen Sensors har en virtuell motsvarighet, Sensors Duplicates, se Figur 33. Denna är en exakt kopia av Sensors, med skillnaden att variablerna i Sensor Duplicates uppdateras av operationer istället för av de fysiska sensorerna. Hur detta fungerar förklaras i nästa avsnitt.

Sensors_Duplicates

	Name	Data type	Offset	Start value	Retain	Visible in ...	Setpoint	Comment
1	Static							
2	Fixture_Floor_Rear	Bool	0.0	false	☑	☑	☐	
3	Fixture_Floor_Front	Bool	0.1	false	☑	☑	☐	
4	Fixture_Roof_Rear	Bool	0.2	false	☑	☑	☐	
5	Fixture_Roof_Front	Bool	0.3	false	☑	☑	☐	
6	Fixture_Side_Left	Bool	0.4	false	☑	☑	☐	
7	Fixture_Side_Right	Bool	0.5	false	☑	☑	☐	
8	Fixture_Cylinder_Up_Front	Bool	0.6	false	☑	☑	☐	
9	Fixture_Cylinder_Down_Front	Bool	0.7	false	☑	☑	☐	
10	Fixture_Cylinder_Up_Rear	Bool	1.0	false	☑	☑	☐	
11	Fixture_Cylinder_Down_Rear	Bool	1.1	false	☑	☑	☐	
12	R2_POS	DInt	2.0	0	☑	☑	☐	0=Hemma, 1=Flexlink, 2=Fixtur, 3=Kör
13	R3_POS	DInt	6.0	0	☑	☑	☐	0=Hemma, 1=Flexlink, 2=Table1, 3=Kör
14	R4_POS	DInt	10.0	0	☑	☑	☐	0=Hemma, 1=Table1, 2=Fixtur, 3=Kör
15	R5_POS	DInt	14.0	0	☑	☑	☐	0=Hemma, 1=Fixtur, 2=Table2,3=Kör

Figur 33: Bild på databasen Sensors Duplicates i TIA Portal

Den fjärde och sista databasen kallas för Modelled. Där placeras de egenmodellerade variabler som saknar en fysisk motsvarighet att jämföras med. För PSL placeras här bland annat variabler som modellerar vad robotarna håller i för bildel, se Figur 34.

9	H3_PRODUCT	DInt	8.0	0	☑	☑	☐	0=Ingen,1=Golv,2=Tak,3=VänsterSida,4=HögerSida
10	H4_PRODUCT	DInt	12.0	0	☑	☑	☐	0=Ingen, 5=LegoMotor, 6=LegoVäxellåda, 7=LegoSäten
11	R2_PRODUCT	DInt	16.0	0	☑	☑	☐	0=Ingen, 1=Golv, 2=Tak, 3=VänsterSida, 4=HögerSida
12	R3_PRODUCT	DInt	20.0	0	☑	☑	☐	0=Ingen,5=LegoMotor,6=LegoVäxellåda,7=LegoSäten
13	R4_PRODUCT	DInt	24.0	0	☑	☑	☐	0=Ingen, 5=LegoMotor, 6=LegoVäxellåda, 7=LegoSäten
14	R5_PRODUCT	DInt	28.0	0	☑	☑	☐	0=Ingen, 1=Färdig bil

Figur 34: Bild på databasen Modelled i TIA Portal

Exempel: Resursen robot R2 har modellerats med två variabler: 'R2_POS' som innehåller robotens position och 'R2_PRODUCT' som innehåller vilken produkt roboten håller i för tillfället. R2_POS sätts av roboten och är tillförlitlig nog för att betraktas som en sensor för den fysiska verkligheten. R2_POS placeras därmed i databasen Sensors, variabeln kan ses i Figur 32. Samtidigt modelleras en dublett av 'R2_POS', som placeras i databasen Sensors Duplicates, denna variabel kan ses i Figur 33. Variabeln R2_PRODUCT är helt igenom modellerad då sensorer för detta saknas. R2_PRODUCT placeras därmed i databasen Modelled, variabeln kan ses i Figur 34.

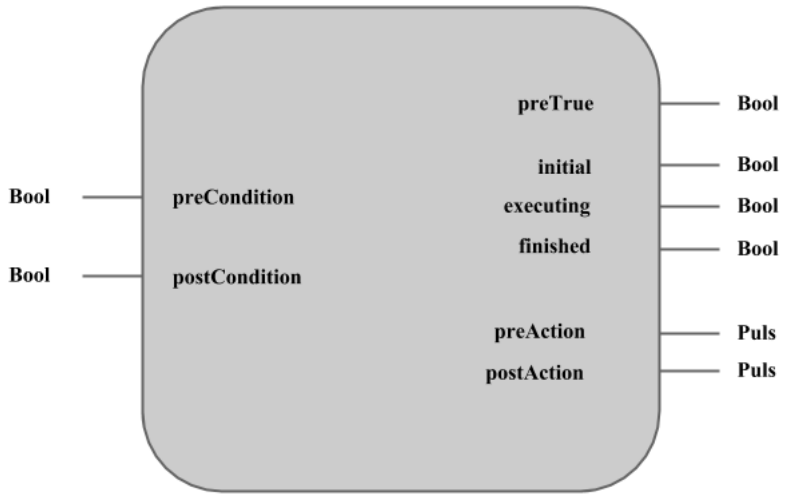
5.2 Operationer i produktionssystemet

Ett system uppbyggt av operationer möjliggör enligt teoriavsnitt 2.3 en flexibel och robust styrning av automatiserad produktion. I PSL har varje operation som uppgift att föra sammansättningen av miniatyrbilen ett steg framåt. En operation kan där till exempel utgöras av att en robot ska förflytta sig från sitt hemmaläge till ett läge vid FlexLinkbanan för att hämta upp en bildel. För att täcka in alla moment i tillverkningen av miniatyrbilen har totalt 46 olika operationer skapats. I detta avsnitt beskrivs hur operationen har implementerats i Siemens programvara TIA Portal och hur denna implementation fungerar. Avsnittet avslutas sedan med ett exempel på en implementerad robotoperation.

5.2.1 Den implementerade operationen

I PSL:s PLC-programvara Siemens TIA Portal fanns vid detta projekts inledning redan ett operationblock implementerat. Detta block har uppdaterats för att bättre uppfylla syftet om effektivare återstart.

En skiss på hur det uppdaterade operationsblocket ser ut i Siemens TIA Portal kan ses i Figur 35. Detta operationsblock fungerar som mall för instansiering av operationer i programmet. Samtliga instanser erhåller identiska gränssnitt för styrning och kontroll av operationen. Denna mall har använts till alla 46 operationer.



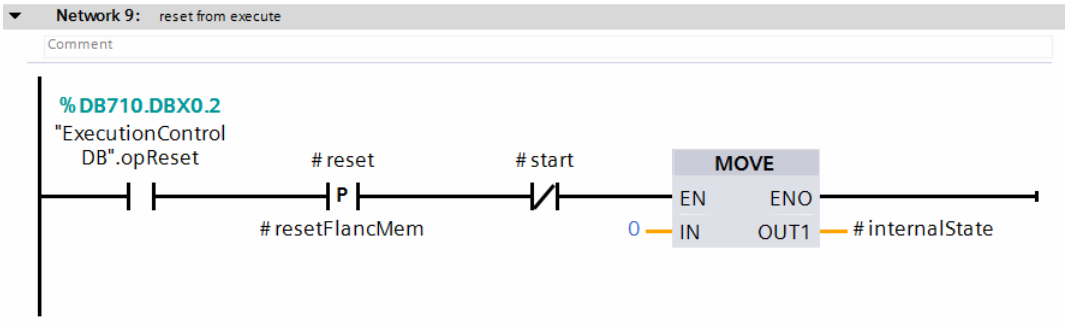
Figur 35: En skiss av det nya operationsblocket

Operationsblocket har de två insignalerna *preCondition* och *postCondition* samt de sex utsignalerna *preTrue*, *initial*, *executing*, *finished*, *preAction* och *postAction*. En operations start- och slutvillkor benämns här *preCondition* och *postCondition* och operationens initial-, exekverings- och sluttillstånd benämns *initial*, *executing* och *finished*. De start- och sluthandlingar som utförs när en operation övergår från initial- till exekveringstillstånd respektive exekverings- till sluttillstånd benämns *preAction* och *postAction*. Utsignalen *preTrue* berättar om operationens startvillkor är uppfyllt och därmed om operationen går att starta eller ej.

Inbyggt i blocket finns också en start- och en återställningsvariabel som används för att starta och återställa operationen. I teoriavsnittet 2.3 om Operationen finns inte sådana variabler, utan en operation startas automatiskt så fort startvillkort är uppfyllt. I detta projekt sker inte detta automatiskt utan när startvillkoret är uppfyllt behöver man även ettställa startvariabeln för operationen i fråga. Dessa variabler används för att kunna styra operationerna från ett användargränssnitt.

När operationens startvillkor uppfylls och operationen befinner sig i sitt initialtillstånd blir utsignalen preTrue sann. Operationen kan då startas genom ettställning av dess startsignal. Om så sker övergår operationen från initial- till exekveringstillstånd. Samtidigt skickas en puls som triggar operationens starthandling. I PSL utgörs starthandlingen vanligen av sändning av ett programnummer till en robot som då börjar röra på sig. Slutvillkoret är i dessa fall att roboten meddelat att uppgiften slutförts. När slutvillkoret uppfyllts övergår operationen från sitt exekveringstillstånd till sitt sluttillstånd. Vid denna övergång skickas en puls som aktiverar operationens sluthandling. En vanlig sluthandling är att uppdatera påverkade variabler för produktionssystemets tillstånd, i exemplet den virtuella variabeln som representerar robotens läge. När operationen är i sitt sluttillstånd kan operationen slutligen återställas till sitt initialtillstånd genom ettställning av operationens återställningsvariabel.

En anledning till varför operationsblocket reviderades var för att det ursprungliga inte gick att föra från exekveringstillståndet tillbaka till initialtillståndet. I ett scenario där en operation fastnade i sitt exekveringstillstånd för att slutvillkoret aldrig blev uppfyllt krävdes en omkompilering av hela programkoden. En vanligt exempel i PSL var då robotarna av någon anledning inte uppfattade det tillskickade programnumret. I dessa fall påbörjades aldrig de rörelser som krävdes för att uppnå operationens slutvillkor och föra operationen till sitt sluttillstånd. I strävan efter effektiv återstart är funktionalitet nu implementerad som möjliggör denna förflyttning. Implementationen innebar ett extra nätverk (network) i operationsblocket logik, se Figur 36. Genom detta kan operatören återställa en fastkörd operation, men först efter att denne ettställt en särskild global variabel (opReset). För att förhindra att operationen startar om direkt efter återställningen finns även ett krav på att dess startvariabel inte får vara ettställd.



Figur 36: Återstart från exekverings till initialtillstånd

Exempel: Ett exempel på hur operationsblocket används för robot R2 kan ses i Figur 37. Denna operation förflyttar robot R2 från sitt hemmaläge till ett läge vid FlexLinkbanan (R2_HomeToFlexlink). I figuren syns tre olika nätverk:

I nätverk 1 syns en instans av det standardiserade operationsblocket. Här kan man se att startvillkor och slutvillkor utgörs av en variabel vardera. Startvillkoret (R2_POS == 0) kräver att roboten R2 ska vara i sitt hemmaläge innan operationen tillåts starta. Slutvillkoret (R2_POS == 1) betyder att roboten R2 ska vara vid fixturen.

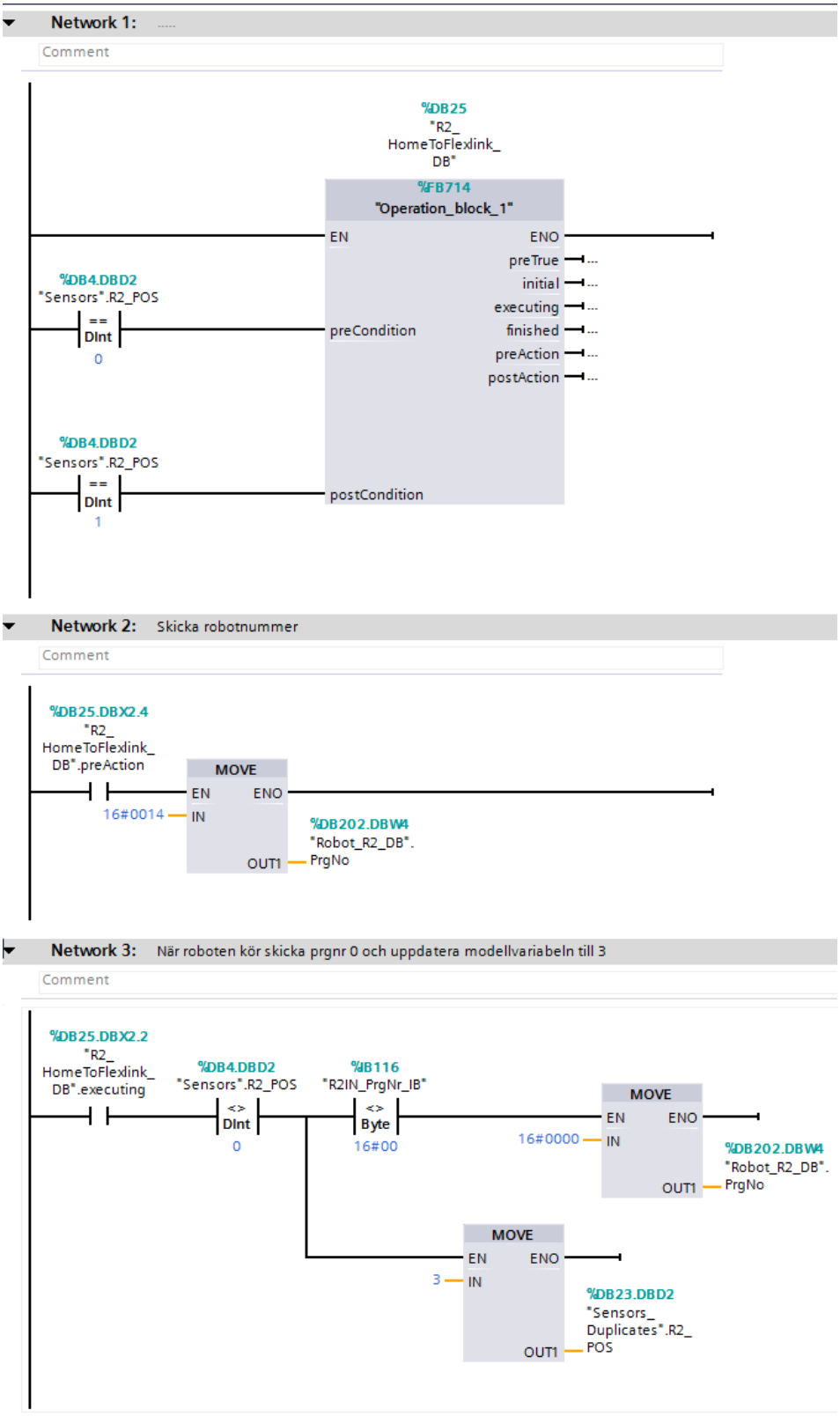
I nätverk 2 sker starthandlingen. Denna aktiveras av en puls när operationens tillstånd övergår till exekveringstillståndet. I detta fall skrivs ett robotnummer till roboten genom ett MOVE-block där värdet på ingångens variabel förs över till utgångens variabel (IN skickas till OUT1). Roboten kommer då att börja röra på sig.

I nätverk 3 skickas en nolla till roboten. Denna behövs för att roboten inte ska börja om på samma rörelse när den är klar. Den virtuella positionvariabeln för R2_POS uppdateras även till 3 vilket betyder att den virtuella modellen förväntar sig att roboten befinner sig i sitt exekveringstillstånd. Se Figur 33 för robotarnas olika tillstånd.

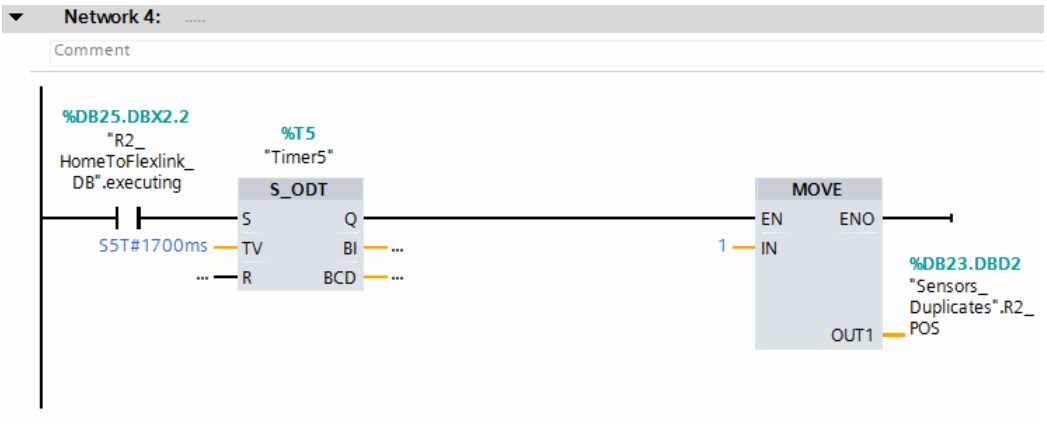
När en rörelse sker i produktionssystemet kommer variablerna i Sensors att uppdateras automatiskt i takt med de fysiska sensorerna. Detta gör dock inte de virtuella variablerna i Sensor Duplicates. Dessa uppdateras istället av själva operationen. Vilka variabler i Sensor Duplicates som uppdateras beror på vilken operation som körs. En operation påverkar bara en delmängd av variablerna.

I nätverk 4 uppdateras bara den virtuella positionsvariabeln för R2, se Figur 38. Detta görs genom att en timer aktiveras precis när roboten börjat röra sig. Tiden som timern är inställd på (1,7 sekunder) är aningen längre än den tid det tar för roboten att i verkligheten förflytta sig från hemmaläget till FlexLinkläget. Denna tid har erhållits genom simulering av robotrörelsen i RobotStudio.

I detta exempel på operation finns ingen sluthandling. Om så varit fallet hade ytterligare ett nätverk krävts likt nätverket för starthandlingen.



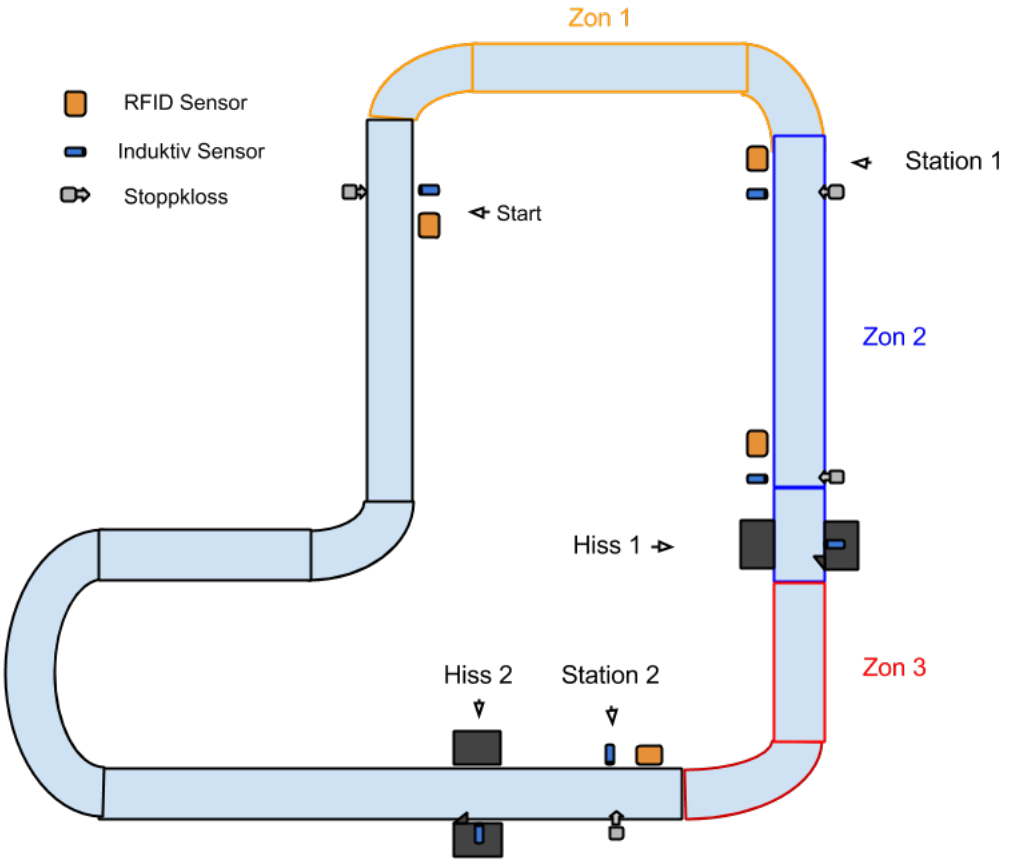
Figur 37: En operation i TIA Portal



Figur 38: Fortsättning av operationen i TIA Portal

5.2.2 Styrning utan operationer: den autonoma FlexLinkbanan

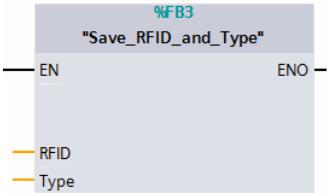
FlexLinkbanan styrs till största del utan operationer. Operationer används där enbart för att skicka in delar och för för att skicka tillbaka stativen från hissarna när robotarna plockat delarna. Stoppen och hissarna är inte styrda av operationer utan sköter sig själva. Detta gör att operatören inte behöver blandas in lika mycket i styrningen. Om ett stopp ska öppnas eller en hiss höjas beror på vilken bildel som passerar den aktuella RFID-sensorn. I Figur 39 ses en skiss på FlexLinkbanan med sensorer, stopp och hissar som är relevanta för styrningen.



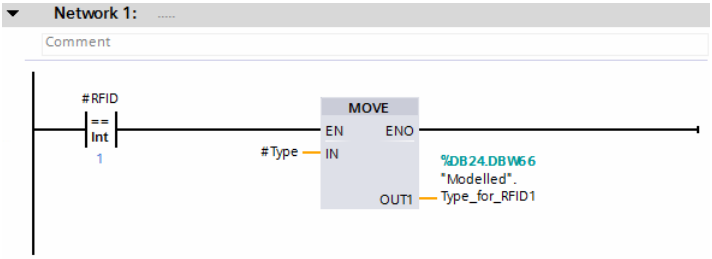
Figur 39: Skiss på flexlinkbanan med stationer och zoner

Vid starten placeras bildelarna manuellt på paletter av en operatör. Startplatsens stoppkloss styrs av operationer. Det finns en operation för varje enskild bildel som kan skickas in på Flex-Linkbanan. Vid en sådan operation kommer RFID- numret på paletten som bildelen är monterad på, samt vilken typ av bildel det är att sparas undan. Detta görs i operationernas starthandlingar precis innan delen skickas in.

För att spara undan RFID-numret och typen har (likt operationsblocket) ett funktionblock skapats, se Figur 40. Detta block har två insignaler: *RFID* och *Type*. På insignalen *RFID* skickas den aktuella palettens RFID-nummer in genom avläsning av RFID-sensorn vid starten och på insignalen *Type* skickas en siffra mellan 1-7 in beroende på vilken operation det är som aktiveras. Funktionsblocket innehåller lika många nätverk som det finns paletter i PSL. I varje nätverk sparas typen undan i en specifik variabel beroende på inmatat RFID-nummer. I Figur 41 ses ett nätverk inuti funktionblocket. Om RFID-numret som skickades in exempelvis är lika med ett sparas typen i variabeln Type_for_RFID1. Genom att det finns lika många nätverk som paletter i PSL kan stativen användas av vilken helst palett som operatören finner lämplig.



Figur 40: En bild på funktionsblocket som sparar RFID och Typ

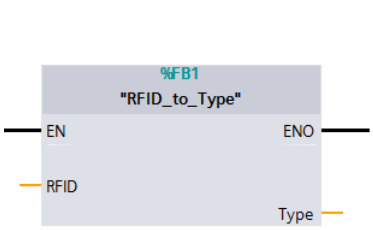


Figur 41: En bild inuti funktionsblocket som sparar RFID och Typ

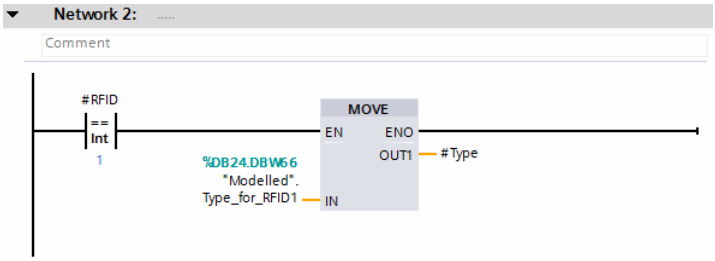
För att förhindra att delarna krockar med varandra kan bara en bildel vara inne i en viss zon. Dessa zoner modelleras av variabler som ettställs och nollställs när en bildel skickas in eller ut ur en zon. Vart de olika zonerna ligger kan ses i Figur 39.

I Figuren är även två stopp som här kallas för stationer utritade, station 1 och station 2. På stationerna finns en RFID-sensor, en induktiv sensor och en stoppkloss. Alla bildelar kommer att stanna vid dessa stationer. Beroende av vilken typ av bildel som står vid en station ska den antingen hissas upp i hissen framför eller åka vidare. Om det står en karossdel (alltså inte lego) vid station 1 ska denna del hissas upp i hiss 1. Om det istället är en legodel som står vid stationen ska den sändas vidare till station 2. Vid varje station sker också en kontroll av att det inte redan är en bildel upphissad innan delen skickas vidare.

För att avgöra vilken typ det är som står vid stationerna har ytterligare ett funktionsblock skapats, se Figur 42. Detta block har en insignal *RFID* och en utsignal *Type*. Om RFID-nummer skickas in erhålls vilken typ av bildel paletten bär. Även detta funktionsblock innehåller lika många nätverk som det finns paletter att tillgå i PSL. I Figur 43 visas en bild på ett av nätverken i funktionblocket. Om det RFID-nummer som skickas in exempelvis är lika med ett skickas värdet som ligger i variabeln Type_for_RFID1 till utgången.



Figur 42: En bild på funktionsblocket som tar emot RFID och ger ut typ



Figur 43: En bild inuti funktionsblocket som tar emot RFID och ger ut typ

5.3 Feldetektering

När det uppstår ett fel i ett automatiserat produktionssystem är det viktigt att operatören tidigt får reda på detta. Om inte kan ytterligare operationer utföras och därmed skada material eller människor samt försvåra den kommande återstarten.

För att upptäcka fel utnyttjas teorin i avsnitt 2.5 om feldetektering. Databaserna Sensors och Sensors Dublicates jämförs kontinuerligt av ett särskilt funktionsblock. Då ett fel uppstår i cellen kommer den fysiska modellen att avvika från den virtuella modellen (som avbildar det förväntade beteendet). När detta sker utlöses ett alarm som upplyser operatören om att modellerna är ur synk. För att alarmet inte ska utlösas för enkelt tillåts modellerna att avvika under en begränsad tid. Tiden är via en timer satt till 1,5 sekund.

Figur 44 visar en bild på databaserna i Siemens TIA Portal när produktionssystemet befinner sig i sitt initialtillstånd. Variablerna i databaserna stämmer då överens, vilket kan ses genom att värdena i kolumnen "Monitor value" är samma i båda databaserna. Figur 45 visar en bild på databaserna efter att robot R2 försökt att montera ett golv på fixturen men tappat golvet på vägen dit. Detta har orsakat en avvikelse mellan de fysiska och de virtuella variablerna för fixturens två golvsensorer, vilket kan ses på de två översta raderna. De fysiska variablerna har falsk status medan de virtuella har sann status.

Sensors

	Name	Monitor value
1	Static	
2	Fixture_Floor_Rear	FALSE
3	Fixture_Floor_Front	FALSE
4	Fixture_Roof_Rear	FALSE
5	Fixture_Roof_Front	FALSE
6	Fixture_Side_Left	FALSE
7	Fixture_Side_Right	FALSE
8	Fixture_Cylinder_Up_Front	TRUE
9	Fixture_Cylinder_Down_Front	FALSE
10	Fixture_Cylinder_Up_Rear	TRUE
11	Fixture_Cylinder_Down_Rear	FALSE
12	R2_POS	0
13	R3_POS	0
14	R4_POS	0
15	R5_POS	0

Sensors_Duplicates

	Name	Monitor value
1	Static	
2	Fixture_Floor_Rear	FALSE
3	Fixture_Floor_Front	FALSE
4	Fixture_Roof_Rear	FALSE
5	Fixture_Roof_Front	FALSE
6	Fixture_Side_Left	FALSE
7	Fixture_Side_Right	FALSE
8	Fixture_Cylinder_Up_Front	TRUE
9	Fixture_Cylinder_Down_Front	FALSE
10	Fixture_Cylinder_Up_Rear	TRUE
11	Fixture_Cylinder_Down_Rear	FALSE
12	R2_POS	0
13	R3_POS	0
14	R4_POS	0
15	R5_POS	0

Figur 44: Databaserna Sensor och Sensors Duplicates i deras initialtillstånd

Sensors

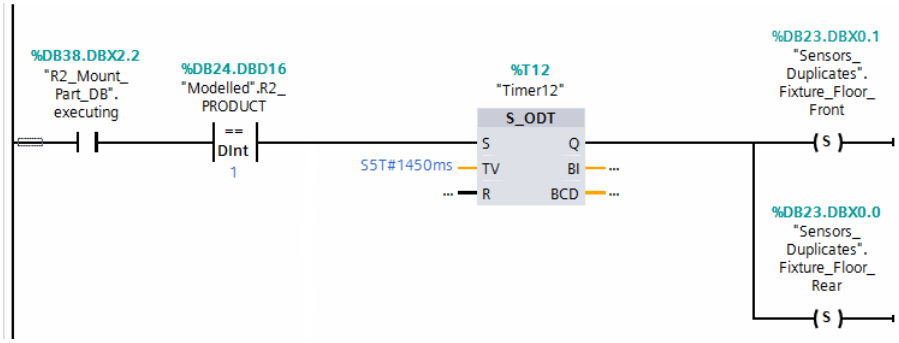
	Name	Monitor value
1	Static	
2	Fixture_Floor_Rear	FALSE
3	Fixture_Floor_Front	FALSE
4	Fixture_Roof_Rear	FALSE
5	Fixture_Roof_Front	FALSE
6	Fixture_Side_Left	FALSE
7	Fixture_Side_Right	FALSE
8	Fixture_Cylinder_Up_Front	TRUE
9	Fixture_Cylinder_Down_Front	FALSE
10	Fixture_Cylinder_Up_Rear	TRUE
11	Fixture_Cylinder_Down_Rear	FALSE
12	R2_POS	2
13	R3_POS	0
14	R4_POS	0
15	R5_POS	0

Sensors_Duplicates

	Name	Monitor value
1	Static	
2	Fixture_Floor_Rear	TRUE
3	Fixture_Floor_Front	TRUE
4	Fixture_Roof_Rear	FALSE
5	Fixture_Roof_Front	FALSE
6	Fixture_Side_Left	FALSE
7	Fixture_Side_Right	FALSE
8	Fixture_Cylinder_Up_Front	TRUE
9	Fixture_Cylinder_Down_Front	FALSE
10	Fixture_Cylinder_Up_Rear	TRUE
11	Fixture_Cylinder_Down_Rear	FALSE
12	R2_POS	2
13	R3_POS	0
14	R4_POS	0
15	R5_POS	0

Figur 45: Databaserna Sensor och Sensors Duplicates då dessa avviker från varandra

Det är, som tidigare nämnts, produktionssystemets operationer som står för uppdateringen av de virtuella variablerna. Figur 46 visar tre nätverk som uppdaterar berörda virtuella variabler, tagna ur en operation där robot R2 monterar ett golv i fixturen. När operationen befinner sig i sitt exekveringstillstånd (roboten rör på sig) och robot R2 har fått i uppdrag att montera ett golv (R2_PRODUCT == 1) startas en timer. Efter 1,45 sekunder uppdateras de virtuella variablerna i fixturen som berör golvet (Fixture_Front_Floor och Fixture_Front_Rear sätts sanna).

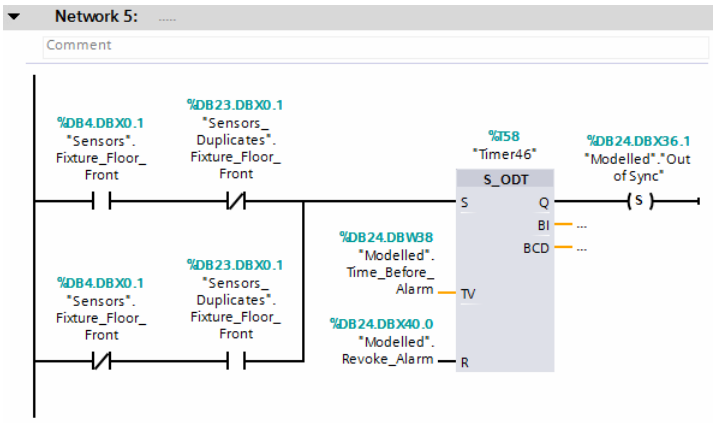


Figur 46: En timer som fördröjer uppdateringen av en virtuell variabel i en operation

Om allt går som förväntat i produktionssystemet ska också de fysiska variablerna för fixturens golvsensorer i databasen Sensors gå sanna efter lika lång tid. Om roboten istället tappar golvet under rörelsen uppstår en avvikelse mellan de fysiska och de virtuella variablerna i likhet med Figur 45.

De parvisa jämförelserna av variablerna i Sensors och Sensor Duplicates görs rad för rad i var sitt nätverk. Vid en avvikelse i någon av jämförelserna startas en timer. Tiden på denna timer är den maximala tid som variablerna tillåts avvika från varandra. Fördröjningen behövs eftersom modellerna aldrig kommer att uppdateras exakt samtidigt. Om felet kvarstår till dess att timern räknat ned utlöses ett larm.

I Figur 47 visas ett nätverk som jämför den fysiska och den virtuella variabeln för fixturens främre golvsensor. Detta görs med hjälp av en XOR-grind (Exklusiv Eller). Om variablerna har olika värden blir signalen till timern sann. Om olikheten sedan står sig tills timern räknat ned aktiveras alarmvariabeln Out of Sync.



Figur 47: Feldetektering för fixturens främre golvsensor

Exempel: För robot R2 kan variabel R2_POS från databasen Sensors jämföras mot sin virtuella dublett R2_POS från databasen Sensors Duplicates . Om dessa inte stämmer överens under en sammanhängande 1,5 sekund löses larmet ut och produktionen stoppas.

5.4 Återstart

När ett fel har uppstått i produktionsystemet och ett larm utlösts innebär det att de två tillståndsmodellerna ej längre stämmer överens. En sådan situation kan ses i Figur 45. För att återstarta produktionen från detta felläge behöver modellerna återsynkroniseras. Det kan antingen göras genom korrigering av den verkliga modellen, den virtuella modellen eller en kombination av de båda. Vad som är lämpligt att manipulera beror helt på den situation som har uppstått eller vad operatören önskar göra.

I PSL är det oftast möjligt att återsynkonisera modellerna genom att för hand korrigera produktionssystemet. Detta är dock inte alltid möjligt i tillverkningsindustrin där produkten kan vara tung, kräva hög precision som enbart robotar kan klara av etc. Detta kan i PSL lösas genom manuell körning av robotarnas rutiner med FlexPendanterna så att de utför de nödvändiga rörelserna mot önskat tillstånd.

Alternativt väljer operatören att återsynkronisera modellerna genom korrigering av den virtuella modellen. Det är möjligt att ändra värdena på modellvariablerna så att de stämmer överens med aktuellt fysiskt tillstånd eller det återstartstillstånd som önskas. Detta gör att systemet kan startas om från vilket tillstånd som helst förutsatt att den fysiska modellen anpassas därefter.

Ibland kan det uppstå oförutsedda fel som tvingar kvar en operation i dess exekveringsfas. För att återstarta systemet från ett sådant tillstånd används operationsblockets nya implementering för just detta. Därefter korrigeras modellerna så att dessa stämmer övererens och att produktionen slutligen kan återupptas.

All korrigering av modellerna sköts av operatören utan vägledning. Det är dock aldrig möjligt för operatören att starta upp systemet i farliga tillstånd då operationernas startvillkor först måste uppfyllas.

Exempel: Förutsätt att larmet för robot R2 har utlösts sedan operationen R2_HomeToFlexlink fastnat i sitt exekveringsläge. Den virtuella dubletten R2_POS hävdar då läge 2: Flexlink medan verklighetens R2_POS meddelar läge 3: Exekverande. Återstart kan då genomföras genom nollställning följt av en omstart av operationen. Då modellerna är återsynkroniserade kvitteras sedan larmet genom ett klick på knappen Reset Sync Alarm i Security-vyn.

5.5 Ett intuitivt användargränssnitt

För att förenkla interaktionen mellan operatör och produktionssystem har ett intuitivt användargränssnitt (GUI, Graphical User Interface) utvecklats. GUI:t är webbläsarbaserat och kan alltså köras på valfri enhet med modern webbläsare. För mer information om installation och uppstart av GUI:t, se Appendix C och B. För mer information om de underliggande teknikerna, se Appendix D och E.

Systemets variabler och operationer utgör de minsta byggstenarna i GUI:t och har genomgående ett likartat utseende. Oavsett vy visualiseras en variabel som ett rektangulärt block indelat i en etikett- och en statusdel. Den vänsterjusterade variabeletiketten upptar merparten av variabelblockets bredd. Etiketten överensstämmer med variabelns namn i OPC:n (OLE for process control, se Appendix D), vilket underlättar spårning för utvecklare. Till höger i variabeln återfinns aktuell status på variabeln. Om variabeln är av boolesk typ visas där en brytare som är ställd i antingen läge “true” eller läge “false”. Läge “true” visualiseras med höger kub fylld med ljusgrön bakgrundsfärg och den svarta texten “true” ovanpå, se figur 48a. Läge “false” visualiseras

i sin tur med vänster kub fylld med ljusröd bakgrundsfärg och den svarta texten “false” ovanpå, se figur 48b.

(a) Sann status **(b)** Falsk status

Figur 48: En boolesk variabel med olika status

Om variabeln är av heltalstyp visas istället aktuellt heltal centrerat över de två kuberna i variabelns statusdel. Texten är svart och lagd ovanpå en ljusgrå bakgrundsfärg. Dessutom tillkommer en extra rad där innebörden av det aktuella heltalsvärdet förklaras av en beskrivande etikett, se figur 49. I denna figur går även att se hur den virtuella variabeln i avsnitt **??** visualiseras i användargränsnittet. För att en sådan etikett ska visas krävs dock att det aktuella värdet definierats upp, se instruktioner senare i detta avsnitt.

Figur 49: En heltalsvariabel

Somliga variabler, så som sensorer och aktuatorer, är endast avsedda för läsning. Samtidigt finns det andra variabler vars syfte är att styra funktioner genom skrivningar. I GUI:t åtskiljs dessa genom att olika listor försetts med ett attribut “toggle-able” satt till sant eller falskt. Om “toggle-able” är satt till falskt för en variabellista får användaren ingen återkoppling då denne för muspekaren över eller klickar på listans variabler. Ett klick växlar då inte heller värde på variabeln. Om däremot “toggle-able” är satt till sant för en variabellista får användaren återkoppling då denne för muspekaren över och klickar på listans variabler.

(a) Med muspekaren förd över **(b)** Vid musklick

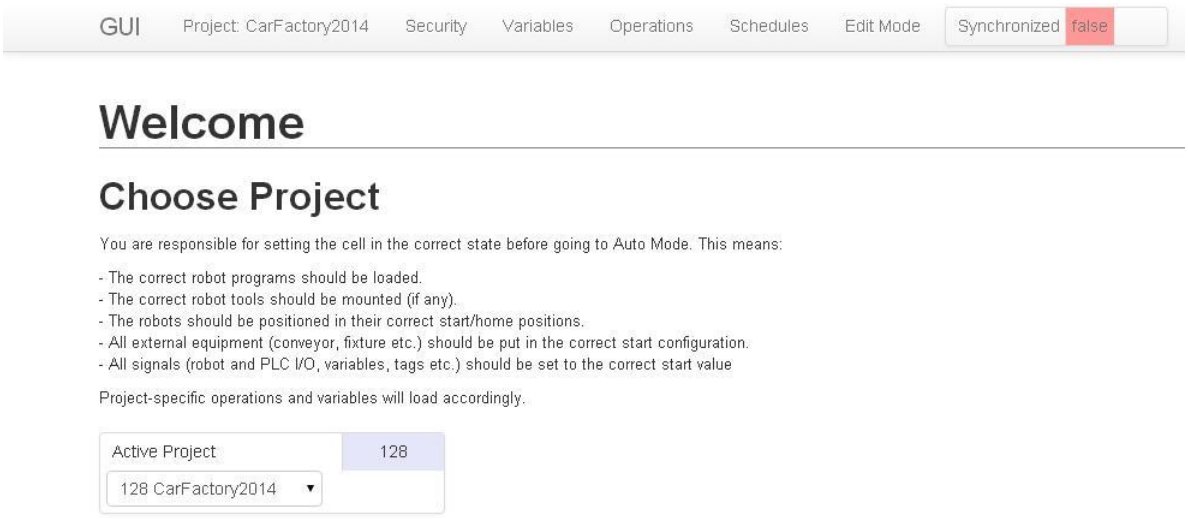
Figur 50: Musinteraktion för en omkopplingsbar variabel.

Om variabeln är av boolesk typ växlas samtidigt dess status till att bli motsatt den dåvarande. För en heltalsvariabel fälls istället en rullgardinsmeny ned med valbara värden. Ett klick på önskat värde innebär då att variabeln status växlas till avsedd status.

Figur 51 visar vyn “Project”, vilken är den första som användaren möter vid uppstart av GUI:t.

Tanken med ett projektval är att flera projekt ska kunna samexistera i samma produktionssystem på ett säkert sätt. Funktionen kommer till sin särskilda rätt i en experimentcell som den i PSL. Själva logiken sköts av en särskild programvalsfunktion i PLC:n. Beroende på valt projekt (motsvarad av en siffra) genomlöps där olika listor med funktionsanrop, vilket garanterar att två projekt inte körs samtidigt. Då användaren inte vill att något projekt ska exekveras, exempelvis vid manuell körning med robotarnas FlexPendanter, kan projektväljaren sättas till “0: None”.

Överst i Figur 51 ovan syns också navigeringsfältet som är bestående genom alla vyer. Navigeringsfältet består av sju klickbara länkar:



Figur 51: Startsidan i det nya GUI:t.

(a) Val av ett annat projekt. **(b)** Ett annat projekt valt.

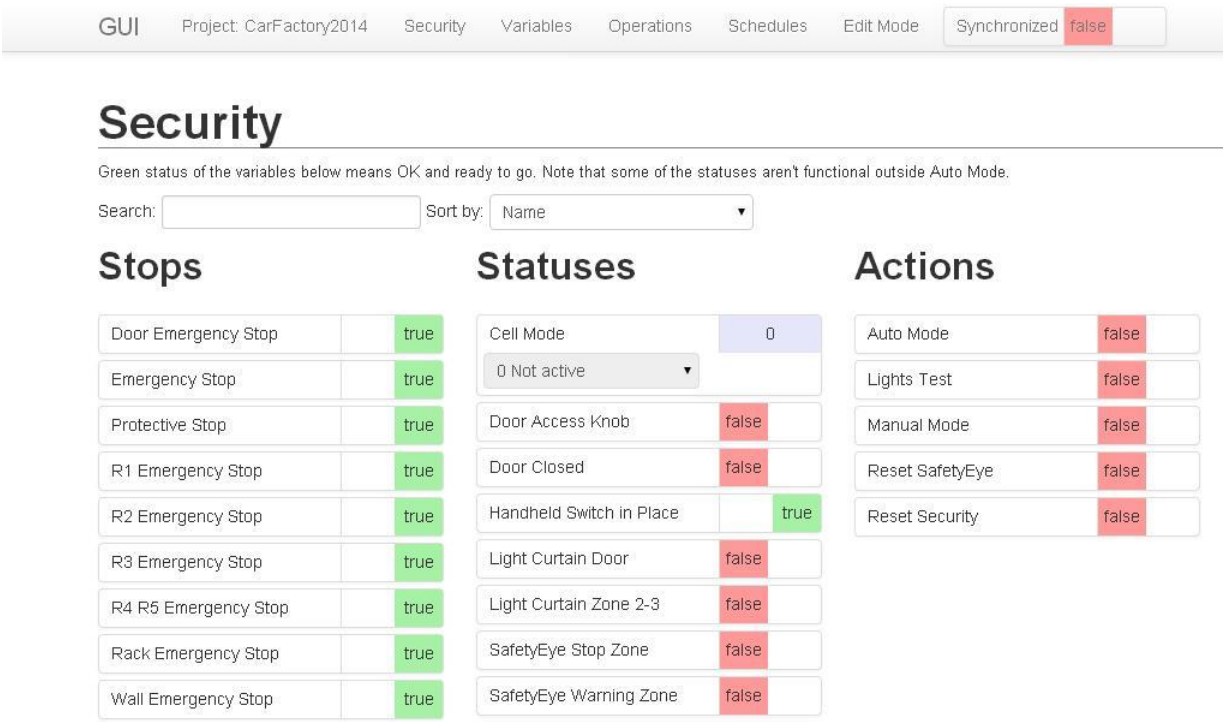
Figur 52: Projektval

Länk	Beskrivning
GUI	Öppnar sida med hjälp, krediteringar och import/export-funktionalitet.
Project: CarFactory	Öppnar projektväljaren som också är GUI:ts startsida. Aktuellt projekt visas efter kolonet.
Security	Öppnar sida med säkerhetsvariabler.
Variables	Öppnar sida med projektspecifika variabler.
Operations	Öppnar sida med projektspecifika operationer.
Schedules	Öppnar sida med projektspecifika operationssekvenser.
Edit Mode	Växlar GUI:ts redigeringsläge av och på.

Vyn "Security", Figur 53, listar alla variabler som rör produktionssystemets säkerhetsanordningar. Dessa är uppdelade i tre kolumner:

Kolumn	Påverkbar	Beskrivning
Stops	Nej	Listar nöd- och skyddsstopp och om de är utlösta eller ej.
Statuses	Nej	Listar status från olika säkerhetsanordningar så som dörrar, ljusbommar etc.
Actions	Ja	Listar möjliga handlingar för säkerhetssystemet så som nollställningar och lägesval.

Säkerhetsvariablerna är uppdefinierade så att sann status (och därmed ljusgrön färgläggning)

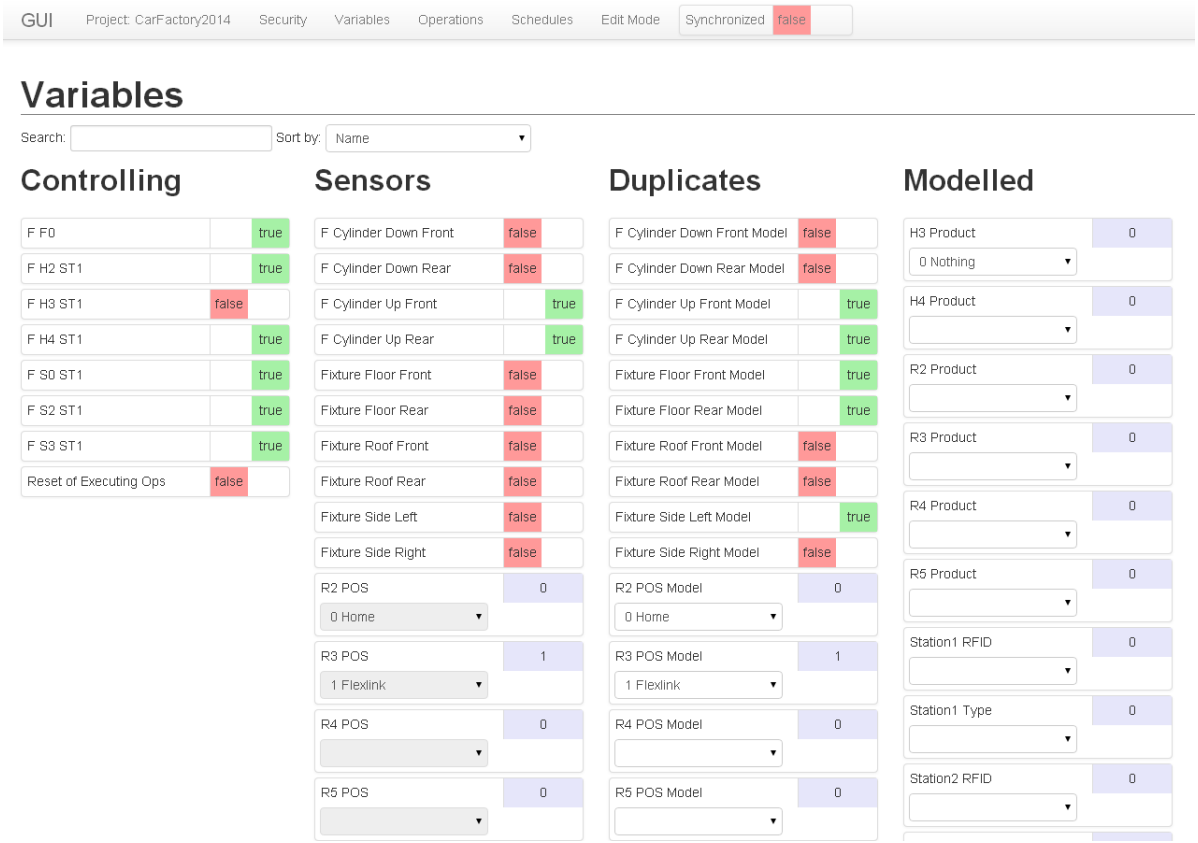


Figur 53: Vyn "Security" som listar variabler för cellens säkerhetssystem.

motsvarar ett körbart tillstånd. En Stopp- eller Statusvariabel i läge "falskt" tyder således på ett fel som måste åtgärdas innan tillverkning kan (åter)startas i cellen.

Vyn "Variables", Figur 54, listar alla projektspecifika variabler. Dessa är uppdelade i fyra kolumner:

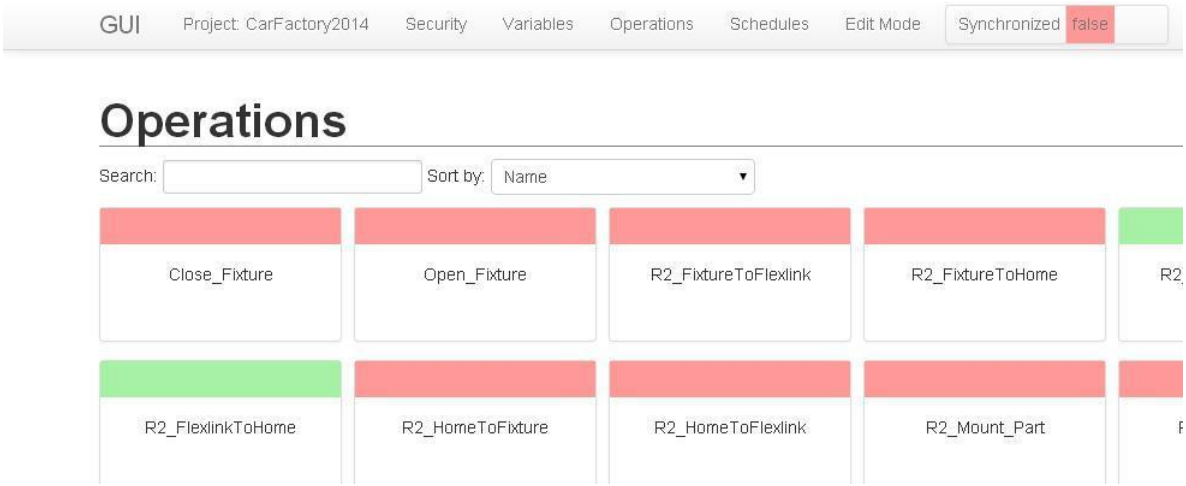
Kolumn	Påverkbar	Beskrivning
Controlling	Ja	Listar projektspecifika, styrande variabler.
Sensors	Nej	Listar projektspecifika variabler för sensorer.
Sensor Duplicates	Ja	Listar projektspecifika, modellerade dubletter för sensorvariablerna.
Modelled	Ja	Listar projektspecifika, helt modellerade variabler.



Figur 54: Vyn "Variables" som listar projektspecifika variabler.

En styrande variabel styr något fysiskt i ett produktionssystem. Det kan exempelvis bestå i att öppna och stänga en ventil eller ett relä. En sensorvariabel återspeglar status för en fysisk sensor i ett produktionssystem, där sensorn exempelvis kan kontrollera kontakt, temperatur eller en RFID-kod. De modellerade variablerna existerar bara internt i PLC:n och täcker upp för önskvärda tillståndskontroller där det saknas fysiska sensorer.

Vyn "Operations", figur 55, listar alla projektspecifika operationer.



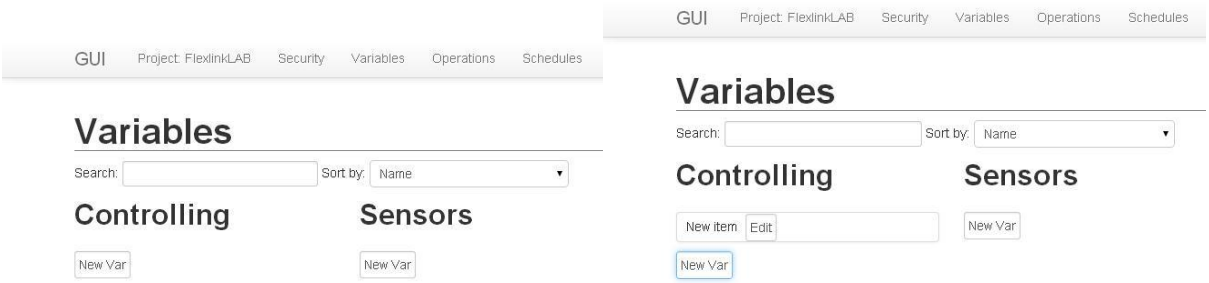
Figur 55: Vyn "Operations" som listar projektspecifika operationer.

Operationernas utseende är inspirerad av skisser som finns tillgängliga i Dr. Bengtssons avhandlingar och kan ses i Figur 2. Basen utgörs av ett rektangulärt block vilket i höjdled är indelat i tre delar. Delarna motsvarar operationens initial-, exekverings- och sluttillstånd. Operationerna listas i ett rutnät om fem i bredd och med nödvändigt antal rader. Operationernas tillstånd visualiseras på följande fyra sätt:

(a) Init, preTrue sann **(b)** Init, preTrue falsk **(c)** Exekveringstillstånd **(d)** Sluttillstånd

Figur 56: En operation i de olika tillstånden.

Knappen "Edit Mode", Figur 57a, försätter GUI:t i redigeringsläge. Med redigeringsläget aktivt synliggörs en knapp "Create New" för varje variabellista och en knapp "Edit" för varje variabel.



(a) Vyn "Variables" med redigeringsläget påslaget. **(b)** En ny, tom variabel skapad i vyn "Variables"

(c) En ny, tom variabel satt i redigeringsläge **(d)** Redigering av variabelnamnet

Figur 57: Skapande av en ny variabel

Ett tryck på "Create New" innebär att en ny variabel med unikt rad-id, namnet "New Variable", en koppling till det aktuella projektet och av för kolumnen specificerad typ skapas i databasen. Omedelbart efter skapandet uppdateras vyn med den nya variabeln inkluderad, se Figur 57b.

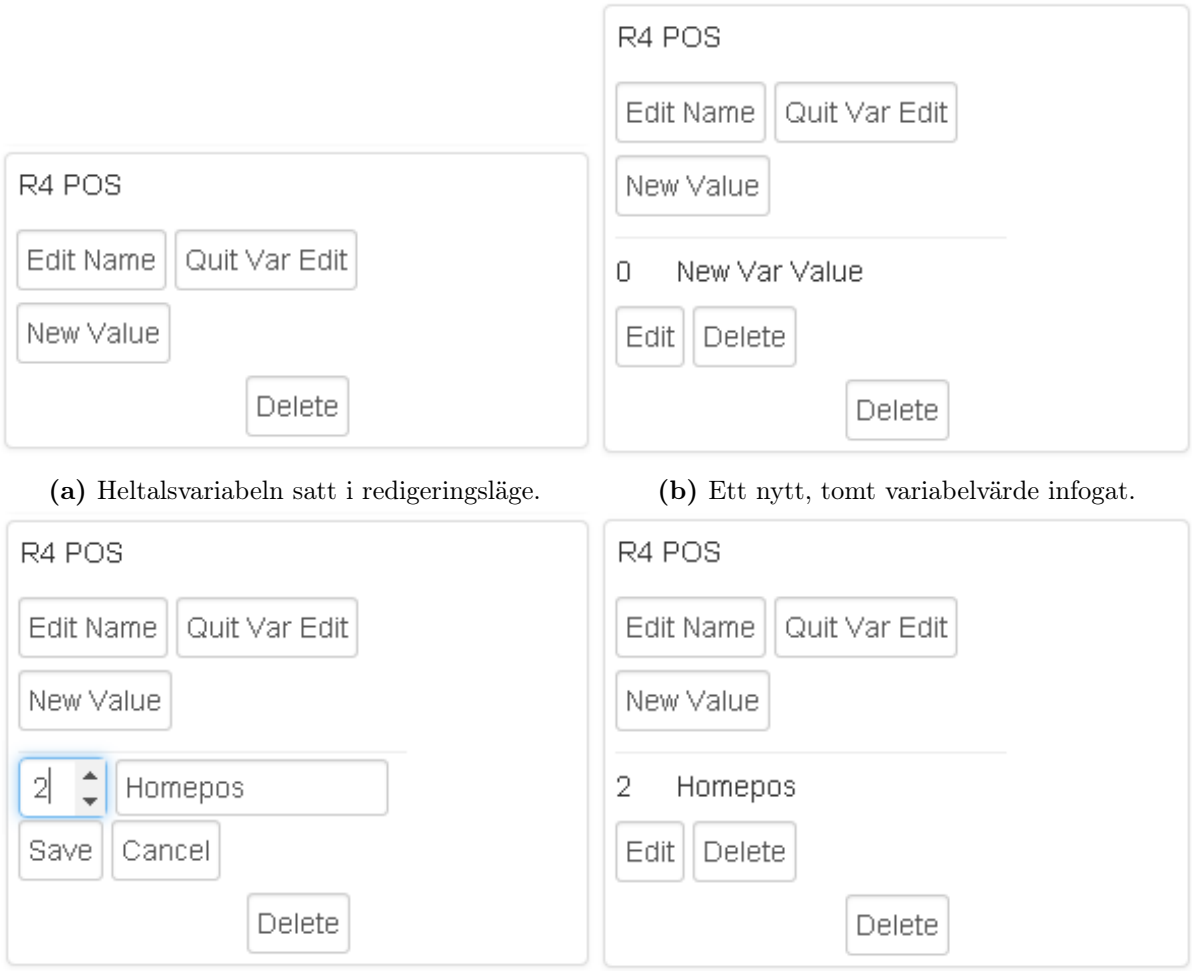
Ett klick på knappen "Edit" vid den nya variabeln öppnar den för vidare redigering, se Figur 57c.

Ett klick på knappen "Edit Var Name" öppnar vidare ett textfält som möjliggör redigering av variabelns namn, se Figur 57d. Namnet ska anges likadant som i OPC:n för att sammankoppling ska ske.

Ett klick på knappen "Save" uppdaterar därpå listan med det nya variabelnamnet. Samman-

koppling mot OPC:n sker omedelbart och GUI:t detekterar sedan automatiskt om variabeln är av boolesk typ eller heltalstyp. Om variabeln är av typen heltal synliggörs en ny knapp, "New Value", se Figur 58a.

Ett klick på "New Value" infogar ett nytt variabelvärde med unikt rad-id, en koppling till den aktuella variabeln, heltalsvärde 0 och namnet "New Value" i databasen. Omedelbart efter skapandet uppdateras vyn med det nya variabelvärdet inkluderat, se Figur 58b.



(a) Heltalsvariabeln satt i redigeringsläge.

(b) Ett nytt, tomt variabelvärde infogat.

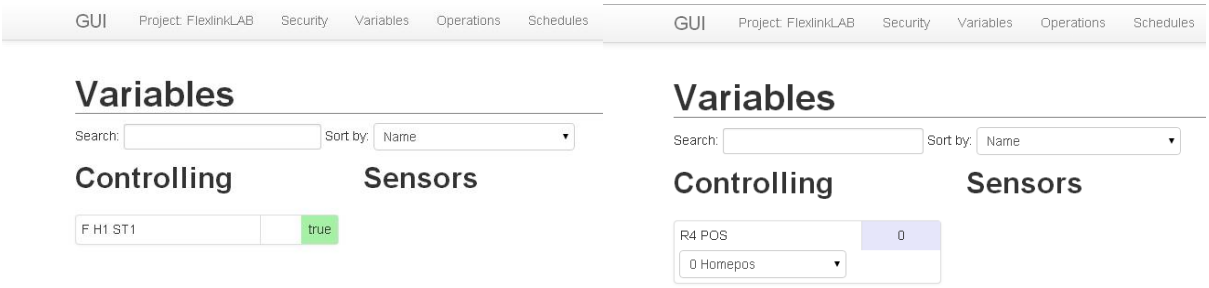
(c) Ett nytt variabelvärde under definiering.

(d) Ett nytt variabelvärde definierat.

Figur 58: Definiering av ett nytt värde för en heltalsvariabel.

Genom ett tryck på knappen "Edit Value" kan önskat heltalsvärde och etikett ställas in och sedan sparas genom ett tryck på knappen "Save", se Figur 58c och 58d.

När redigeringen av variabeln är klar avslutas den genom ett tryck på knappen "Quit Var Edit". Då slutligen redigeringen av hela vyn är klar avaktiveras redigeringsläget genom ett återställande tryck på knappen "Edit Mode" i navigeringsfältet. Den nya variabeln framträder då enligt någon av figurerna 59a eller 59b beroende på om den är av boolesk- eller heltalstyp.



(a) Boolesk variabel **(b)** Heltalsvariabel

Figur 59: Två fungerande, nyskapade variabler.

6 Diskussion

I följande kapitel tolkas och diskuteras resultaten och dess reliabilitet jämfört mot mål, syfte och tidigare forskning. Här diskuteras också potentiella framtida arbeten.

6.1 Syfte, centrala resultat och metod

En utmaning inom industriautomation idag är att återstarta produktion när ett fel har uppstått. Om felet inte upptäcks direkt kan det fortplanta sig genom systemet. Detta gör återstartsfasen svår. Ofta behöver hela linor stoppas och startas om för att produktionen ska komma igång igen (Andersson, Lennartsson och Fabian 2010). Detta projekt har haft som syfte att utveckla en flexibel programstruktur med tillhörande användargränssnitt som förenklar styrning av, tidigt upptäcker fel i och underlättar återstart av automatiska produktionssystem.

Under projektets gång har mötesformen Scrum använts för kommunikation mot projektägaren, avstämning av mål och framsteg och uppdelning av arbetsuppgifter. Därpå har visualiseringsverktyget Kanban använts för synliggörande av projektets status.

Arbetet inleddes med studier i aktuell forskning. Därpå implementerades utvalda ideer i ett produktionssystem vid Production Systems Laboratory (PSL), Chalmers.

Projektet har resulterat i en identifiering av produktionssystemets resurser och programmering av dess förmågor. Systemets variabler har delats upp i en fysisk och en virtuell modell. Den fysiska modellen ska återspegla produktionssystemets verkliga tillstånd medan den virtuella modellen ska återspegla produktionssystemets förväntade tillstånd. Den fysiska modellen uppdateras främst av sensorer medan den virtuella modellen främst uppdateras av operationer.

6.2 Utvärdering av den valda metoden

I projektets tidiga stadier var gruppen av uppfattningen att samarbete med andra projektgrupper i PSL skulle förekomma. Med tiden utvecklades dock projektet till att bli allt mer självständigt.

Scrum hölls inte efter helt då samarbetet inom gruppen var stort och redovisningar om utfört arbete skedde kontinuerligt, flera gånger per dag. I övrigt passade dock scrummetodiken bra för detta projekt då det handlade om prototypframtagning utan ett från början väldefinierat tillvägagångssätt.

6.3 Avvägningar för modellerna och dess operationer

Robotarna har programmerats för att genom SetDO-instruktioner ettställa digitala utgångar motsvarande det viloläge som roboten befinner sig i. En heltalsvariabel i PLC:n för robotens position sätts till korrekt värde utifrån detta. Variabeln kan tyckas modellerad, men har faktiskt betraktats som en verklig sensor. Anledningen är att robotens sista instruktion har noggrannheten Fine. Denna innebär att roboten inte sätter DO till 1 innan Fine-instruktionen har slutförts, det vill säga nått sin slutpunkt. På så vis garanterar robotens inbyggda kontroller ett visst mått av verklighetsförankring.

Namn på variabler hålls konsekvent samma genom de båda databaserna “Sensors” och “Sensor Duplicates”. Variablerna kan åtskiljas genom dess hemhörighet i respektive databas, men innebär ändå en viss risk för sammanblandning.

Behovet av en Controlling-databas kan ifrågasättas, då skrivning lika gärna kan göras direkt till den berörda PLC-taggen. Databasen tjänar dock två syften. Dels samlar den alla styrande variabler för aktuellt projekt på ett och samma ställe i PLC:n. Dels tydliggörs hur uppdelning i de olika typerna gjorts när samtliga projektvariabler har hemhörighet i någon av de fyra databaserna Controlling, Sensors, Sensor Duplicates och Modelled.

Att använda både fysiska och virtuella variabler i start- och slutvillkor har sina för- och nackdelar. Fördelen är att det tillåter systemet att utnyttja variabler som saknas i det fysiska systemet. Nackdelen är att det aldrig är helt säkert om dessa villkor är helt uppfyllda eftersom det bara rör sig om börvärden.

FlexLinkbanans paletter antas inte föra med sig en speciell typ av del. På detta sätt undviks låsning till vissa paletter som lätt kan tappas bort eller blandas ihop, vilket rimmar väl med målet om god flexibilitet.

6.4 När operationen passat och inte

Under projektets gång har olika tillvägagångssätt testats för att styra produktionssystemets FlexLinkbana. Det har visats sig att det inte är lika enkelt att styra FlexLinkbanan med operationer som för andra delar i produktionssystemet. Detta beror främst på att FlexLinkbanan består av många olika komponenter (stopp, hiss etc.) som på flera sätt är beroende av varandra. Fel i samspelet mellan komponenterna kan exempelvis få bildelar att fastna bakom en upptagen hiss.

Ett tillvägagångssätt som testades var att styra FlexLinkbanan med enbart operationer. Detta gjorde att operationerna blev små och många. Med så många små operationer blev banan i slutändan mer komplicerad och tidskrävande för operatören att styra än tidigare. I ett scenario där flera bildelar är på ingång på FlexLinkbanan läggs därmed för stort ansvar på operatören att hantera styrningen. Detta är inte önskvärt då projektet vill underlätta styrning för operatören.

Som ett andra tillvägagångssätt utvärderades en variant med betydligt mer omfattande operationer, motsvarande “Skicka del till hiss 2”. Detta förenklade operatörens jobb betydligt, men begränsade samtidigt antalet samtidiga delar på FlexLinkbanan till en enda. Samtidig körning av två operationer hade annars kunnat störa ut varandra utan ett annat angreppsätt i operationens uppbyggnad. Dubbel körning av samma operation är inte heller möjlig rakt av då PLC:ns minnesadresser inte kan utökas dynamiskt och därmed inte tillåter instansiering av operationer under körning.

En tänkbar lösning på instansieringsproblemet var att ta fram en helt ny typ av operation: den asynkrona operationen. Ett särdrag för den asynkrona operationen skulle i sådana fall vara att den går att starta flera gånger. Detta förutsätter någon form av förallokering av minnesutrymmen som sedan kan tas i bruk då behovet uppkommer under körning. Maximalt antal samtidigt körbara instanser måste hur som helst fastställas i designfasen då de blir oföränderliga efter kompilering. I fallet FlexLinkbanan hade visserligen inte denna avvägning varit särskilt svår, då mängden stopp begränsar antalet samtidiga delar till runt fyra stycken. En annan aspekt är felkontrollen. Då en asynkron operations exekveringstid blir olika lång beroende på FlexLinkbanans tillstånd i övrigt kan inte felkontroll angripas på samma vis som en ordinarie operation. Vad som möjligen kan kontrolleras är kortare steg i den asynkrona operationen som helt säkert inte ska påverkas av något yttre, så som transport av en bildel från ett stopp till nästa.

Det slutliga valet för styrningen av FlexLinkbanan föll som bekant inte på den asynkrona operationen utan på en mer autonom styrning av varje enskilt stopp. Operationernas roll begränsas då till att sända in en viss del för att sen genast bli återställningsbara. Feldetektering kan utvecklas lik den diskuterades för den asynkrona operationen, dock skulle en sådan sakna koppling till en

viss platta, del eller operation. Den autonoma lösningens stora fördel är att instansieringsproblematiken försvinner ur världen. Operatören kan föra in valfritt antal plattor med olika delar utan att hindras av annat än att den första FlexLinksektionen måste vara ledig.

6.5 Användargränssnittet

Det framtagna modulära GUI:ts funktionalitet underlättar styrning och återstart genom den tydliga visualiseringen. Prestandajämförelse mot etablerade lösningar som Siemens WinCC ger klar fördel till det nyutvecklade, webbläsarbaserade GUI:t. Upp- och omstart tar bara några sekunder jämfört med minuter med WinCC. Styrning från surfplatta, smart-TV eller smartphone är ett annat område där GUI:t står över all konkurrens.

Specificering av variabler och operationer innebär tyvärr mycket dubbeljobb för utvecklaren i den befintliga lösningen. Då PLC:n av säkerhets- och tillförlitlighetsskäl ska stå för huvuddelen av villkorstestningen måste allt först specificeras i PLC:ns utvecklingsprogramvara. Därpå måste OPC:n specificeras för alla variabler som ska skickas vidare. Slutligen ska samma information också matas in i GUI:t. Jämfört med att arbeta i etablerade gränssnittsprogram så som Siemens WinCC är skillnaden visserligen inte jättestor. Siemens WinCC (för skapande av HMI:s, Human Machine Interfaces) kräver ju till exempel skapande av så kallade HMI-tags som mellansteg till användargränssnittet. Ett ultimat sätt att komma runt detta problem skulle vara att utvecklaren specificerar operationen i GUI:t, vilken därpå autogenererar nödvändig PLC-kod och injicerar denna i PLC:n. En sådan lösning förutsätter dock att PLC:n tillåter sådan kodimport antingen helautomatiskt eller via manuellt ingripande av utvecklaren. Alternativt skulle operationerna kunna överföras automatiskt från PLC:n. Detta förutsätter istället automatiskt export från PLC:n, en funktion vilken för närvarande dessvärre är lika höljd i dunkel som den automatiska importen.

Vid en första anblick verkade SQLites filbaserade lagring som en underlättande väg till direktkopiering av databasen mellan datorer. Detta visade sig snart felaktigt, då rättighetsproblem förhindrade skrivning från den nya värden. Efter idoga försök till lösning fick en kringlösning skapas i form av import/export-funktionen i About-vyn. Slutanvändare som inte har läst de bifogade instruktionerna och försöker att flytta hela mappens innehåll rakt upp- och ned kommer med all sannolikhet att stöta på problem därav.

Målet har varit att användargränssnittet ska vara så intuitivt som möjligt. Detta är dessvärre inte helt fallet i vyn Security där flera variabler för nöd- och skyddstopp är OK då de har sann status. “Emergency Stop: true” motsäger variabelns gröna färg som, helt riktigt signalerar att allt är som det ska. Etiketten kunde försetts med ett mer logiskt namn, så som “Emergency Stop OK”. Alternativt kunde säkerhetslogiken i PLC:n kodats om eller GUI:t försetts med en funktion för signalinvertering.

Intuitiviteten har dock gynnats av val av konsekventa färger och form för operationer och variabler och att överflödig information hålls undan från användaren när den inte behövs. Noterbar är också den redundans som föreligger i variablernas statussignalering. Texten true eller false ackompanjeras med grön eller röd färg och har olika position beroende på om variabeln för tillfället är sann eller falsk.

6.6 Feldetektering

Genom den konstanta sökningen efter skillnader mellan den fysiska och den virtuella modellen upptäcks de flesta fel snabbt. I funktionen för feldetektering finns en timer utplacerad som tillåter viss fördröjning av alarmet när en avvikelse har upptäckts. Denna kommer alltid att behövas då

produktionssystemets operationer inte kan klockas på skanncykelnivå. Idag är tiden till utlösning avvägd till 1,5 sekund. Om denna tid är välavvägd har inte hunnit utvärderas vid tidpunkten för denna rapport, men är beroende av hur väl operationerna modellerar verkligheten. RobotStudios simuleringstider, vilket de nuvarande modelleringarna är baserade på, behöver inte nödvändigtvis vara realiserbara i verkligheten.

Feldetekteringen har delats upp i flera nätverk för att möjliggöra flera olika alarmvariabler. Just nu används dock bara en enda samlande alarmvariabel för alla fel som kan uppstå, vilket ålägger användaren att genom GUI:t finna det faktiska felet. Detta fungerar hjälpligt för produktionscellen i PSL då den inte är så komplex, men försvårar felsökning i större system.

6.7 Återstart

Under den återstartsprocess som används i detta projekt läggs allt ansvar på operatören. Det medför att en viss kunskap om cellen är ett måste. Det är möjligt att sätta cellen i vilket tillstånd som helst vilket också innebär att risk finns för att starta upp cellen i tillstånd där allt låser sig. En ytterligare risk som finns är att operatören tappar bort sig och inte lyckas ändra till önskat tillstånd då det inte finns nån vägledning under återstartfasen. I PSL är risken för detta ganska liten men i större och mer komplexa system kan detta vara ett större problem. Ett ytterligare alternativ som har diskuterats i gruppen är att utföra återstarten genom manuell exekvering av operationer.

Värt att nämna är också frågan om hur mycket det ska vara tillåtet att manipulera den virtuella modellen. Kanske skulle det endast vara tillåtet om systemet är med och hjälper till.

6.8 Vidare arbete

Ett naturligt nästa steg är utveckling av fler funktioner i GUI:t. Operationerna bör visualiseras med start- och slutvillkor samt start- och sluthandlingar. Vidare är det angeläget att en funktion för sammansättning av och kontroll av förlopp för sekvenser av operationer (SOP:ar) tas fram. Innan en sådan funktion finns på plats kan inte en bil byggas per automatik, åtminstone inte genom ren operationsstyrning från GUI:t.

Avvikelser mellan den fysiska och virtuella modellen skulle kunna synliggöras bättre i GUI:t och på så sätt snabbare lokaliseras. Idag går endast ett alarm varpå operatören får hitta avvikelsen på egen hand. Detta fungerar i ett mindre produktionssystem som PSL men kan vara betydligt svårare i ett system med ett större antal variabler.

FlexLinkbanan bör förses med mekanismer för feldetektering. Vidare forskning kring den asynkrona operationer kan där visa sig lönande.

Slutligen har detta projekt inte behandlat beräkning av lämpliga återstartstillstånd. I dagsläget är det operatören själv som avgör från vilken punkt som systemet ska återstartas, vilket i komplicerade produktionssystem kan vara en krävande uppgift. Automatisk beräkning av återstartstillstånd passas därmed helt vidare som ett lämpligt område att realisera praktiskt.

7 Slutsats

Produktionssystemet har avbildats som en fysisk och en virtuell modell. Den fysiska modellen består av variabler som uppdateras av verkliga sensorer. Den virtuella modellen består istället av variabler som uppdateras av operationer. Då ett fel uppstår i produktionen utvecklar sig inte den fysiska modellen som den virtuella modellen. Modellerna är då, med andra ord, osynkroniserade.

Genom konstant jämförelse mellan den fysiska och den virtuella modellen detekteras snabbt avvikelser mellan de två modellerna. En avvikelse av sådant slag utlöser ett larm till operatören.

Kodstrukturen är uppbyggd av operationer vilket gör valet av produktionsupplägg flexibelt. Förändringar av operationerna kan enkelt göras genom omdefiniering av dess startvillkor. Det är också möjligt att lägga till nya operationer i ett senare skede.

För att underlätta styrning och återstart av produktionssystem har nämnd funktionalitet implementerats i ett nytt användargränssnitt. Detta uppmärksammar operatören på larm och visualiserar skillnader i modellerna när ett fel inträffat. Därpå kan operatören modifiera den fysiska eller den virtuella modellen för att återsynkronisera modellerna. Användargränssnittet visar också vilka av operationerna som är möjliga att starta.

Styrning av produktionssystem är inte längre låst till en specifik dator. Tack vare webbteknologi kan det nyutvecklade användargränssnittet köras på valfri enhet med en modern webbläsare. Det är också möjligt att lägga till och redigera både operationer och variabler direkt i användargränssnittet under körning. Detta ger en operatör möjlighet att förändra gränssnittet även om denne inte är insatt i hur den underliggande koden fungerar.

Samtliga utvecklade funktioner har inte hunnit testas i praktiken ännu. Huruvida den utvecklade kodstrukturen och användargränssnittet underlättar återstart av produktionssystem kvarstår därmed att verifera.

Framtida arbete bör också ägnas åt beräkning av lämpliga återstarttillstånd samt utveckling av en funktion för sammansättning av sekvenser av operationer (SOP:ar) i användargränssnittet.

Referenser

Andersson, K., B. Lennartsson och M. Fabian (2010). ”Restarting Manufacturing Systems; Restart States and Restartability”. I: *IEEE Transactions on Automation Science and Engineering* 7.3, s. 486–499.

Andersson, Kristin m. fl. (2012). *Generation of Restart States for Manufacturing Systems with Discarded Workpieces*. URL: `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5234175` (hämtad 2014-05-19).

Axelsson, Mattias (2013). *Vad är Scrum?* URL: `http://www.happiness.se/artiklar/vad-ar-scrum` (hämtad 2014-04-15).

Beck, A. och P. Gohner (dec. 2010). ”Generation of optimization proposals for electrical energy analysis of industrial automation systems”. I: *Energy Conference and Exhibition (EnergyCon)*. Manama, s. 114–119.

Bengtsson, Kristofer (2012). ”Flexible design of operation behavior using modeling and visualization”. Diss. Göteborg: Chalmers University of Technology, Signals och Systems.

Bengtsson, Kristofer och Bengt Lennartsson (nov. 2013). ”Flexible specification of operation behavior using multiple projections”. I: *IEEE Transactions on Automation Science and Engineering* 11.2, s. 504–515.

Bergagård, Patrik (dec. 2013). ”Calculating Restart States for System Modeled by Operations Using Supervisory Control Theory”. I: *Machines* 1, s. 116–141.

Davidor, Yuval (1991). *Genetic Algorithms and Robotics: A Heuristic Strategy for Optimization*. World Scientific.

Groover, Mikell P. (2007). *Automation, production systems, and computer-integrated manufacturing*. 3. utg. Prentice Hall Press.

Hopcroft, John E., Rajeev Motwani och Jeffrey D. Ullman (2000). *Introduction to Automata Theory, Languages, and Computation*. 2. utg. Pearson Education.

Lee, John D. (2012). *Human factors and ergonomics in automation design*. 4. utg. John Wiley och Sons.

Levitt, Theodore (maj 1983). ”The globalization of markets”. I: *Harvard Business Review*.

Pan, Zengxi (2012). *Recent progress on programming methods for industrial robots*. URL: `http://www.sciencedirect.com/science/article/pii/S0736584511001001` (hämtad 2014-05-19).

Roth, Mattias, Jean-Jacques Lesage och Lothar Litz (2011). *The concept of residuals for fault localization in discrete event systems*. URL: `http://www.sciencedirect.com/science/article/pii/S0967066111000396` (hämtad 2014-05-13).

A Uppstart av tillverkningscellen i PSL

Uppstart av tillverkningscellen i PSL med tillhörande robotar, verktyg och programvaror utförs i sju steg:

1. Starta upp GUI:t med dess tillhörande tjänster, se Appendix B.

2. Vrid nyckeln på teknikskåpets gavel till körläget.

3. Sätt cellen i Auto Mode under fliken Security i GUI:t. Pulsa ev. variablerna för Security och SafetyEye Reset.

4. Kvittera blinkande blå tryckknappar i cellen genom ett tryck på vardera av dem. Se till att dörren till cellen är stängd och att dess vridbrytare 'Request Door Access' är ställd i rätt läge (och därmed inte lyser). Avsluta med ett tryck på grön knapp. Cellen bör nu gå upp i Auto samtidigt som tryckluftsystemet trycksätts.

Montage och demontage av verktygen för R3 och R4 kräver samtidig inblandning av två operatörer: en vid roboten och en vid dess FlexPendant.

5. R3:ans 16 utgångar består av 8 stycken 1-till-2-ventiler. Varje 1-till-2-ventil har alltid en av dess två utgångar öppna, 1 eller 2. Varje ventil styrs av två variabler. Ventilen ställs i läge 1 eller 2 genom pulsning till den variabel som är knuten till önskad utgång. Respektive variabel måste nollställas direkt efter ettställning för att inte blockera de omvända operationerna.

DO_16_1: Tryckluft för fastlåsning av verktyget
DO_16_2: Avstängning av ovanstående
DO_16_3: Tryckluft för stängning av gripklon
DO_16_4: Avstängning av ovanstående
DO_16_5: Tryckluft för öppning av gripklon
DO_16_6: Avstängning av ovanstående

Observera att en stängning av gripklon med signal DO_16_3 måste följas av en avstängning med signal DO_16_4 för att öppning av gripklon med signal DO_16_5 ska fungera som tänkt. Detsamma gäller givetvis omvänd ordning.

6. R4:an är enklare att manövrera då pulsning inte tillämpas. Här är följande signaler aktuella:

R4_Tool_R 0/1: Tryckluft för fastlåsning av verktyget av/på
R4_Rivet_On 0/1: Tryckluft till sugkopparna av/på

7. Bilens delar förs in via FlexLinkbanan. Koden är okänslig för vilken palett som skickas in. Ändå är det bara en palett med stativ på vilken plåtdelarna kan föras in. Fler stativ finns, men då de skiljer sig för mycket åt måttmässigt riskerar robot R2 att misslyckas med plockningen av delar från dessa. Den fungerande paletten med stativ förses med olika moduler beroende på vilken del som ska föras in. De delar som fungerar och hör samman är uppmärkta med förklarande text.

B Uppstart av användargränssnittet och dess nödvändiga tjänster

Uppstart av GUI:ts komponenter utförs i fyra steg:

1. Uppstart av OPCTalkers serverklient på den dator som är ansluten till PLC:n. Detta utförs genom ett dubbelklick på OPCTalkers skrivbordsgenväg. Ett kommandofönster öppnas där OPCTalker meddelar att servern startats och på vilken port.

2. Uppstart av SequencePlanner på den dator som ska agera webbserver. Detta utförs på samma sätt som med OPCTalkers serverklient, genom ett dubbelklick på SequencePlanners skrivbordsgenväg. Om så önskas kan SequencePlanner köras på samma dator som OPCTalkers serverklient. SequencePlanner öppnar ett kommandofönster och matar ut några uppstartsmeddelanden. Snart bör också kontakt etableras mellan SequencePlanner och OPCTalkers serverklient vilket utlöser en snabb utskrift av registrerade variabler och operationer i båda fönstren.

3. Sammankoppling av webbserverdatorn med den/de enhet/enheter som användaren önskar styra och övervaka cellen ifrån. En enhet kan utgöras av en surfplatta, mobiltelefon eller bärbar dator; kravet är att enheten är utrustad med en modern webbläsare. Då webbserverdatorn av säkerhetsskäl i regel inte är publikt tillgänglig via Internet måste enheten anslutas till webbserverdatorns nätverk. Detta kan göras antingen genom fysisk uppkoppling (trådbundet/trådlöst till det slutna nätverkets router) eller via VPN (Virtual Private Network, med användarnamn och lösenord över Internet).

4. Öppning av GUI:t på den/de enhet/enheter som ska styra och övervaka cellen. Detta utförs genom att användaren öppnar ett webbläsarfönster och besöker adressen till GUI:t. Adressen utgörs av 'webbserverdatorns IP-adress':'inställd webbserverport', där webbservern som standard körs på port 8080. I PSL är den fullständiga adressen således 172.16.205.18:8080. Om GUI:t ska öppnas på den enhet som också hyser webbservern kan adressen istället förenklas till localhost:8080.

C Migrering av användargränssnittet till en ny dator

Flytt eller kopiering av SequencePlanner till en ny dator.

Kopiering av SequencePlanner till en ny dator utförs i ett antal steg:

1. Export av databasinnehåll från en befintlig installation. Först ett klick på ikonen "GUI" i GUI:ts navigeringspanel och därefter på knappen "Export". En fil benämnd backup.db skapas i SequencePlanners exekveringsmapp.

2. Kopiering av de nödvändiga filerna till måldatorn. De filer som krävs är den exekverbara jar-filen (GUI for PSL.jar), den fil som innehåller en backup av databasen (backup.db), uppstartsfilen (startMe.bat) och den mapp som innehåller webbfilerna (C:\gui\SequencePlanner\src\main\web). Hela sökvägen måste behållas i det senare fallet då den är hårdkodad i klassen SequencePlanner.scala.

3. Kontroll av att Java JRE finns installerad på måldatorn. Om så inte är fallet ska Java JRE laddas hem och installeras från java.com.

4. Redigering av filen startMe.bat med valfri texteditor. Justering av sökvägen till måldatorns Java-installation, sen sparning och stängning.

5. Förslagsvis skapande av en skrivbordsgenväg till startMe.bat.

6. Omdöpning av backup.db till backup_to_restore.db.

7. Uppstart av GUI:t enligt instruktioner i avsnittet nedan.

8. Import genom ett tryck på knappen Import under fliken GUI. Variabler och operationer bör då installeras på sina respektive sidor.

D OPC och SequencePlanner

Statusläsning och skrivning till variabler i PLC:n är normalt bara möjligt genom att PLC-tillverkarens egen utvecklingsprogramvara (Siemens Step 7 i PSL) sätts i direkt uppkoppling mot PLC:n. För att råda bot på denna inlåsning kan en så kallad OPC-server användas (Siemens Simatic NET i PSL). Serverprogramvaran möjliggör statusläsning och skrivning från externa datorer genom standarden OPC (OLE for Process Control). För att en variabel ska bli åtkomlig externt behöver den först definieras upp i OPC-servern. Sådana uppgifter utförs med hjälp av en OPC-utforskare (Siemens OPCScout V10 i PSL). Uppdefiniering kräver att användaren känner till databasnummer, offset, bitnummer och antal på den variabel som ska definieras upp. Därtill väljer användaren själv en lämpligt namn som identifierare gentemot klienterna.

OPC-standarden bygger idag på tekniken DCOM vilken är smått föråldrad och också låser standarden till Windows-plattformar. I syfte att möjliggöra mer flexibel kommunikation över TCP/IP (arkitektur för datakommunikation) har Dr. Kristofer Bengtsson, Chalmers utvecklat en klient-serverprogramvara med namnet OPCTalker. OPCTalker är skrivet i Scala (ett programspråk vars applikationer körs i Java-maskiner) och är därmed körbart på de flesta plattformar. OPCTalker saknar grafiskt användargränsnitt, utan matar ut rader med information direkt i en kommandotolk/terminal.

På klientsidan är OPCTalker inbyggd som en del av Dr. Bengtssons SequencePlanner, en programvara för att skapa och hantera variabler, operationer samt sekvenser av dessa. SequencePlanner vet inte automatiskt vilka variabler den ska lyssna efter förändringar på. Därför krävs en andra uppdefiniering utöver den som redan gjorts i OPC-servern. Definitionerna läggs direkt i klassen OPCDefinition.scala som körs vid initieringen av SequencePlanner. En variabel definieras som en 2-tupel:

Varopc(name: String, value: String).

Objektnamn	Typ	Beskrivning
name	String	Variabelns namn
value	String	Adress till variabeln i OPC:n

En operation definieras som en 5-tupel:

OPopc(name: String, preTrue: String, state: String, start: String, reset: String).

Objektnamn	Typ	Beskrivning
name	String	Variabelns namn
preTrue	String	Adress till operationens preTrue-variabel i OPC:n
state	String	Adress till operationens state-variabel i OPC:n
start	String	Adress till operationens start-variabel i OPC:n
reset	String	Adress till operationens reset-variabel i OPC:n

Adresserna till de olika variablerna ovan är en sammansättning av OPC-serverns adress (S7:[S7_Connection_1]) och den aktuella variabelns identifierare. Här följer ett exempel på vardera typen:

Varopc(“R2.HAS_FLOOR”, “S7:[S7_Connection_1]R2.HAS_FLOOR.value”)

En definitionen till en variabel med namnet “R2.HAS_FLOOR” som har adressen “S7:[S7_Connection_1]R2.HA
i OPC:n.

OPopc(“R2_home_to_fixture”, “S7:[S7_Connection_1]R2_home_to_fixture.preTrue”, “S7:[S7_Connection_
“S7:[S7_Connection_1]R2_home_to_fixture.start”, “S7:[S7_Connection_1]R2_home_to_fixture.reset”)

En operation med namnet “R2_home_to_fixture” med OPC-adresser till operationens startvillkors-
, tillstånds-, start- och återställningsvariabler.

Då SequencePlanner sätts i kontakt med OPCTalker-servern ansvarar SequencePlanner för att status på variabler och operationer hålls synkroniserade med dess status i PLC:n. Utöver variabeluppdatering agerar SequencePlanner också som webbserver mot vilken webbapplikationer kan läsa och skriva information. I klassen RequestRouter.scala specificeras vilka URL-adresser som ska finnas och vilken data som ska returneras vid förfrågnignar till dessa adresser. Datan levereras enligt JSON (JavaScript Object Notation) vilket är ett kompakt, textbaserat format för datautbyte. Det lämpar sig väl för AJAX-appar (Asynchronous JavaScript and XML), webbapplikationer med hög grad av interaktivitet. En variabel skickas vidare som 2-tupeln

Var(name: String, value: JsValue)

Objektnamn	Typ	Beskrivning
name	String	Variabelns namn.
value	Boolean	Om variabeln för tillfället är sann eller falsk.

En operation skickas i sin tur vidare som 3-tupeln

OP(name: String, preTrue: JsValue, state: JsValue)

Objektnamn	Typ	Beskrivning
name	String	Operationens namn.
preTrue	Boolean	Om operationens startvillkor utvärderats till sant eller falskt.
state	Integer	Ett heltal motsvarande det tillstånd operationen befinner sig i. 0 = initialtillstånd, 1 = exekverande, 2 = klar.

Här följer ett exempel på vardera typen:

Var(“R2.HAS_FLOOR”, true)

En variabel med namnet “R2.HAS_FLOOR” vilken för tillfället har värdet sann i PLC:n.

OP(“R2_home_to_fixture”, false, 2)

En operation med namnet “R2_home_to_fixture” vars startvillkor för tillfället är utvärderad till falskt och befinner sig i tillstånd 2 = klar.

E Utökningar av SequencePlanner och användargränssnittet

Definiering av nya variabler måste fortfarande göras manuellt, och serverprefixet är detsamma som tidigare (S7:[S7_Connection_1]). Samtliga gamla variabler är bortrensade från OPC-servern till förmån för nya CarFactorys variabler. De nya variablerna är namngivna med vanligt mellanslag istället för understreck med ökad läsbarhet som följd och utan programmatiska komplikationer.

SequencePlanner har bestyckats med en SQL-databas (SQL (Structured Query Language) är ett språk för utförande av databasoperationer). Databasen tillhandahålls tekniskt av SQLite, en fri databashanterare som skapar en fysisk fil för databasens innehåll. Drivrutiner finns tillgängliga för en mängd språk och plattformar, däribland Java och Scala. För detta projekt har SQLite-JDBC valts. Genom hanteringsprogram som exempelvis SQLite Studio kan vid behov tabeller och data ändras manuellt. Om databasfilen sp.db inte existerar skapar SequencePlanner denna automatiskt tillsammans med nödvändiga tabeller. Databasen består av följande tre tabeller:

operations

Kolumn	Beskrivning
id	Unikt rad-ID som tilldelas automatiskt då en ny operation läggs till
name	Namn på operationen
project	ID för det projekt som operationen tillhör

variables

Kolumn	Beskrivning
id	Unikt rad-ID som tilldelas automatiskt då en ny variabel läggs till
type	Typ av variabel, exempelvis controlling, sensor eller modelled
name	Namn på variabeln
project	ID för det projekt som variabeln tillhör

var_values

Kolumn	Beskrivning
id	Unikt rad-ID som genereras automatiskt då ett nytt värde läggs till
var_id	ID för den variabel som värdet tillhör
int_value	Heltalsvärde som ska skrivas till variabeln
name	Etikett för vad heltalsvärdet motsvarar i praktiken

En databas möjliggör dynamisk ändring av data under körning som därefter sparas mellan sessionerna. Istället för den tidigare hårda definieringen av variabler och operationer direkt i Scala-klassen OPCDefinition.scala hämtas numera denna data från databastabellerna variables och operations.

Tillsammans med databasen har två nya JSON-definitioner tillkommit. VarValue nedan motsvarar ett värde som en variabel ska kunna anta tillsammans med en beskrivande etikett. Detta är nödvändigt för att en användare ska veta vilka heltalsvärden som går att skriva till en variabel och vad dessa värden motsvarar i praktiken. Ett exempel är variabeln “Active Project” som istället för en ruta för valfritt heltal 0 - 10000 på detta vis istället kan visa alternativen “Flexlinklab” och “CarFactory”.

VarValue(dbID: Int, intValue: Int, name: String)

Objektnamn	Typ	Beskrivning
dbID	Integer	Värdets unika rad-id i databasen.
intValue	Integer	Heltalsvärde som ska skrivas till variabeln.
name	String	Förklarande etikett till heltalsvärdet.

Den andra nytillkomna JSON-definitionen är AlterDBRow. Denna skrivs från användargränssnittet då användaren vill lägga till, ändra eller ta bort en variabel, ett variabelvärde eller en operation ur databasen. Definitionen består av många objekt, men de är inte alla populerade på samma gång. Hur många som populeras är istället beroende av vilken åtgärd som användaren önskar utföra.

Heltalet project är ett nytt värde som fastställer vilket projekt som variabeln eller operationen tillhör. Värdet möjiggör filtrering av variabler och operationer enligt vilket projekt som för tillfället är valt i användargränssnittet. ID:t ska överensstämma med vad som är programmerat i PLC:ns projektanropare. Projektöverskridande variabler för exempelvis säkerhet är tilldelade projekt-ID 0 (“Inget projekt”) för att vara synliga oavsett projektval.

AlterDBRow(table: String, vartype: String, action: String, opID: Int, varID: Int, name: String, intValue: Int, project: Int)

Objektnamn	Typ	Beskrivning
table	String	Tabell som ska redigeras.
vartype	String	Typ på variabel som ska skapas.
action	String	Åtgärd som ska vidtas. Ex. ’insert’, ’edit’ eller ’delete’.
opID	Integer	Rad-ID för påverkad operation.
varID	Integer	Rad-ID för påverkad variabel.
name	String	Namn som ska skrivas till variabel, operation etc.
intValue	Integer	Heltal som ska skrivas till ett variabelvärde.
project	Integer	ID för det projekt som variabeln, operationen etc. ska tillhöra.

Som en följd av de nytillkomna definitionerna ovan består nu JSON-definitionerna för variablerna och operationerna av fler objekt än tidigare. Variablerna definieras numera som:

Var(dbID: Int, name: String, value: JsValue, vartype: String, varvalues: List[VarValue], project: Int)

Objektnamn	Typ	Beskrivning
dbID	Integer	Variabelns unika rad-id i databasen.
name	String	Variabelns namn.
value	Boolean	Om variabeln för tillfället är sann eller falsk.
vartype	String	Typ av variabel, exempelvis ’controlling’ eller ’modelled’.
varvalues	List[VarValue]	Lista med de värden som kan skrivas till variabeln.
project	Integer	ID för det projekt som variabeln tillhör.

Medan operationerna definieras som:

OP(dbID: Int, name: String, preTrue: JsValue, state: JsValue, project: Int)

Objektnamn	Typ	Beskrivning
dbID	Integer	Operationens unika rad-id i databasen.
name	String	Operationens namn.
preTrue	Boolean	Om startvillkoret för tillfället utvärderas till sann eller falsk.
state	Integer	Heltal för det tillstånd som operationen befinner sig i. 0 = initialtillstånd, 1 = exekverande, 2 = klar.
project	Integer	ID för det projekt som operationen tillhör.

F Användargränssnittets uppbyggnad

I slutet av kedjan återfinns ett grafiskt användargränssnitt (GUI). Detta är utvecklat i JavaScript-ramverket AngularJS. AngularJS är framtaget av Google, släpptes 2009 och utgör basen till flera av Googles populära webbapplikationer. Appar i AngularJS programmeras enligt MVC-principen om separering i modell (Model), vy (View) och kontrollant (Controller). AngularJS kännetecknas också av att variabler i vyn hålls synkroniserade med dess modell utan behov av särskilda metoder för uppdatering. Med andra ord kan en statusändring av en variabel direkt leda till motsvarande ändring i vyn (det användaren ser).

Från GUI:t ansvarar en JavaScript-metod för att via en Angular-tjänst (service) begära data från en URL-adress på webbservern. Datan placeras sen i korrekt Angular-lista, så som $scope.variables[]. Via Angulars $timeout-tjänst itereras denna metod var 500:de millisekund, därmed resulterande i en lika tät uppdatering av variablerna i gränssnittet.

En viktig funktionalitet i AngularJS är möjligheten att skapa egna så kallade direktiv. Då ett direktiv infogas i HTML-koden instanseras en på förhand definierad klass inklusive utseende, beteende och andra egenskaper. För GUI:t har bland annat direktivet “var-list” definierats upp, vilket listar variabler av en viss typ. Variabeltypen som ska visas i en viss lista specifieras till “var-list” genom en tillskrivning till attributet “vartype”. Om variablerna i listan ska vara skrivbara bestäms i sin tur genom att attributet “toggleable” sätts till true eller false. Här följer ett exempel på en instansiering av “var-list”:

<div variable-list vartype=“sensor” toggle-able=“false” >

Vyerna Project, Security och Variables har alla byggts upp med hjälp av detta direktiv.