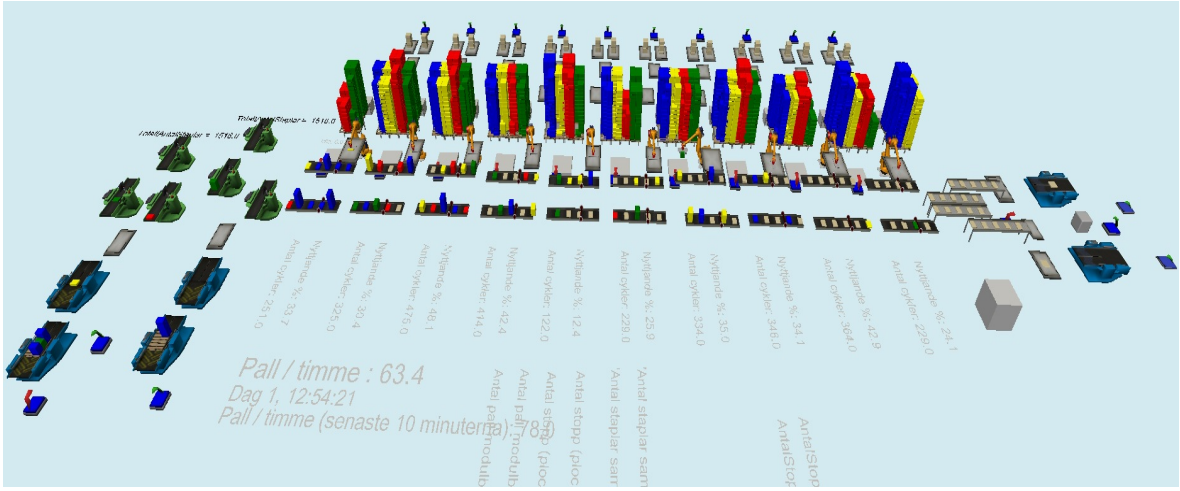
CHALMERS



Simulering av orderplockanläggning

Simulation of production plant

Examensarbete inom högskoleingenjörsprogrammet Mekatronik

Hannes Andreasson
Martin Edviken

Institutionen för Signaler och system
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2016
Examinator: Manne Stenberg

Förord

Detta examensarbete är genomfört på institutionen för Signaler och System på Chalmers Tekniska Högskola och omfattar 15hp per person. Examensarbetet är den avslutande delen av högskoleingenjörsprogrammet Mekatronik, 180 hp. Arbetet som genomfördes på företaget Teamster AB innefattade att uppgradera en befintlig simuleringsmodell till den senaste versionen av simuleringsprogrammet FlexSim samt att verifiera modellen.

Vi skulle vilja tacka företaget Teamster AB och vår handledare på företaget, Magnus Fredriksson, för all hjälp under projektets gång. Dessutom vill vi rikta ett stort tack till vår handledare på Chalmers, Morgan Osbeck, för all hjälp med utformandet av rapporten.
Trevlig läsning!

Hannes Andreasson & Martin Edviken

Sammanfattning

Företaget Teamster AB utvecklade en simuleringsmodell under ett tidigare projekt där de levererade en orderplocksanläggning till Pågen i Malmö. Orderplockanläggningen tillverkar färdiga pallar med bröd efter kundorder. Simuleringsmodellen skapades under projektet för att säkerställa kapaciteten på anläggningen de skulle leverera. Simuleringsmodellen verifierades dock aldrig under projektet. Uppdraget var därför att verifiera modellen samt att uppdatera den till senaste versionen av simuleringsprogrammet för att hålla modellen uppdaterad inför framtiden. Arbetet har skett hos Teamster i ett simuleringsprogram som heter FlexSim. En förstudie av både den fysiska orderplockanläggningen och simuleringsmodellen har gjorts för att sätta sig in i projektet. En självupplärningsfas i simuleringsprogrammet FlexSim har också gjorts då inga tidigare erfarenheter av simuleringsprogram fanns. Arbetet har därefter fortsatt med att uppdatera simuleringsmodellen till senaste versionen av FlexSim samt att verifiera den. Resultatet har blivit en simuleringsmodell som är uppdaterad till senaste versionen samt verifierad mot verklig driftsdata. Den uppdaterade simuleringsmodellen har en bra överensstämmelse med orderplockanläggningen. Ett antal förbättringsåtgärder har applicerats på modellen och Teamster har nu en tillförlitlig simuleringsmodell de kan använda i framtida projekt.

Abstract

The company Teamster AB developed a simulation model during a project were they delivered a production plant to Pågen in Malmö. The production plant manufacture pallets with bread according to the customers specified order. The simulation model was used during the project to calculate the capacity of the production plant that were going to be delivered. A verification of the model was never done during that project. Because of this the assignment was to verify the model and also update it to the latest version of the simulation program to keep the model up to date for the future. The assignment has been done at Teamsters office in a simulation program called FlexSim. A pilot study of both the physical plant and the model was made to gain insight and to be able to accomplish the given project. To be able to work with FlexSim, a self-learning phase were done as well due to the lack of previous experience from simulation programs. The work has continued with the update of the model to the latest version of FlexSim and verifying the model. The result ended in a model that is updated to the latest version of FlexSim and is verified against real production data from Pågen. The simulation model is a good simulation of the real world production plant. The work has been kept within the borders and the delimitation of the project. Some improvements have been applied to the model and Teamster now have access to a simulation model that is reliable and can be used it future projects.

Innehållsförteckning

1. INLEDNING

Detta projekt handlar om att uppdatera en befintlig simuleringsmodell av en orderplockanläggning till senaste versionen av simuleringsprogrammet FlexSim. Efter uppdateringen ska en verifiering av simuleringsmodellen göras.

1.1 Bakgrund

Teamster är ett företag som är verksamma inom automationsbranschen. År 2012 levererade de en komplett orderplockanläggning till Pågens bageri i Malmö. Anläggningen består i huvudsak av 14 robotar, 40 inbanor med bröd, två modulbanor och styrutrustning. Med hjälp av den skapas pallar med brödlådor enligt kundordrar i Pågens affärssystem. Pallarna med bröd transporteras till utlastningsbanor där de hämtas av lastbil för transport till butik eller omlastningsställe. Varje pall innehåller alltså en unik kombination av olika brödlådor.

Vid projektering och under projektet använde Teamster en simuleringsmodell, utvecklad i programmet Flexsim version 5, för att beräkna orderplockanläggningens kapacitet. Simuleringsmodellen är komplex då den innehåller många inparametrar såsom tillgänglighet för ingående maskindelar och egenskaper för inkommande brödpallar. Teamster har för avsikt att använda modellen till liknande orderplockanläggningar och vill kunna utnyttja den förbättrade grafiken och de nya funktionerna i den senaste versionen av simuleringsprogrammet FlexSim.

1.2 Syfte

Syftet med examensarbetet är att uppdatera simuleringsmodellen till FlexSim version 2016 och verifiera denna. Med utgångspunkt från en tidigare simuleringsmodell ska samma funktionalitet uppnås. Vidare ska eventuella möjligheter till förbättringar av modellen analyseras med avseende på verifieringens resultat.

1.3 Avgränsningar

Då migreringen från FlexSim version 5 till version 2016 förväntas vara mycket tidskrävande finns inga krav på att några förbättringsåtgärder appliceras på modellen inom detta arbetets omfattning.

1.4 Precisering av frågeställningen

Krävs det att modellen byggs upp från grunden för att fungera i den nya versionen av FlexSim?
Om simuleringsmodellen behöver byggas upp på nytt, hur mycket underlättar den befintliga simuleringsmodellen uppbyggandet av den nya?
Vilka möjligheter öppnas med de nya funktionerna som finns i den senaste versionen av FlexSim?
Hur bra överensstämmer den nya simuleringsmodellen med orderplockanläggningen?
Vad kan förbättras i simuleringsmodellen?

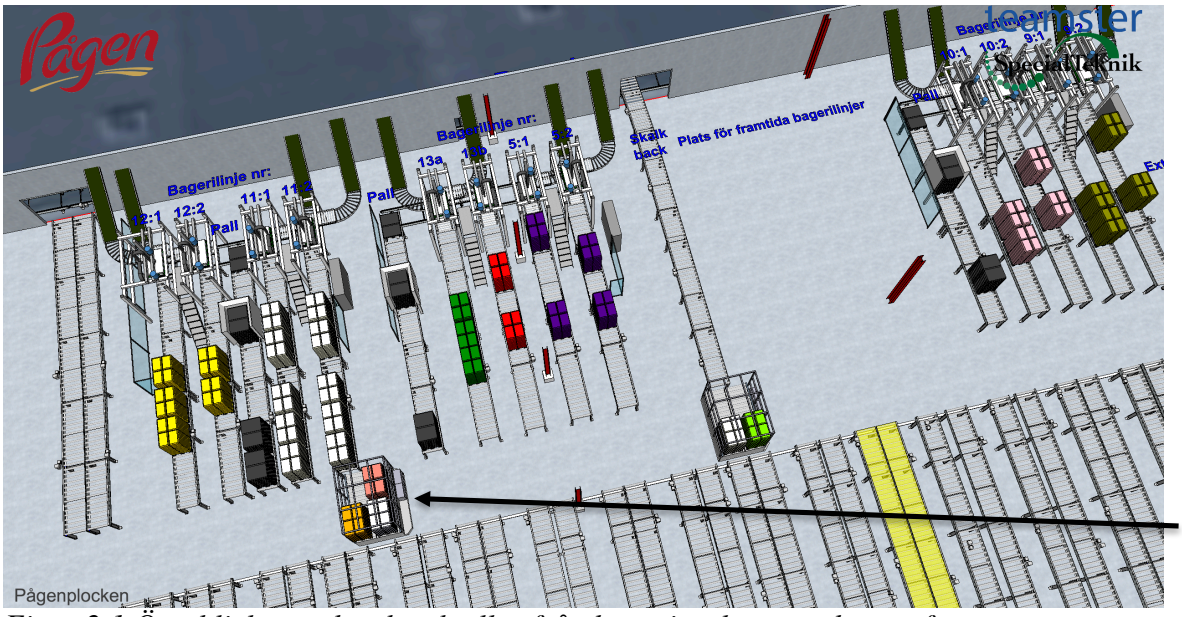
2. TEKNISK BAKGRUND

Här presenteras teknisk data för att få förståelse för processen som ska simuleras samt simulering i allmänhet.

2.1 Pågens orderplockanläggning

Pågen är Sveriges största bageri där bageriet i Malmö tillsammans med bageriet i Göteborg står för all tillverkning [1]. Mellan 1200 – 1700 pallar av Pågens sortiment packas varje dag i bageriet i Malmö. Kunderna lägger dagligen nya beställningar på hur mycket bröd av varje sort de vill ha, vilket gör att varje pall har en unik kombination av antal brödsorter. Detta var både ett komplext och slitsamt hanterande av brödlådor som utfördes av ett 50-tal anställda. År 2011 installerade företaget Teamster en helautomatiserad orderplockanläggning i bageriet i Malmö för att ersätta den manuella packningen. Packlinjen tar upp en yta på 60x60 meter och består av 14 robotar och totalt två km rullbanor [2].

Från bageriet kommer bröd i standardiserade brödlådor och lyfts i grepp om 4 lådor av samma sorts bröd på en lastpall och staplas i önskad höjd. Efter att ha staplats på pall placerar en transfervagn ut pallen på en av de 40 inbanorna till plockrobotarna, se figur 2.1.



Figur 2.1 Överblick över hur brödpallar från bageriet skapas och transfervagn.

Inbanorna har plats för 40 olika sorters bröd men oftast placeras storsäljarna på flera banor. Det är dock endast en sorts bröd per inbana och inte blandat. Vid inbanorna finns det tio robotar som använder fyra inbanor vardera. Tvärs inbanorna och utefter robotraden går två stycken parallella transportbanor, kallade modulbanor, se figur 2.2. På dessa modulbanor placerar plockrobotarna brödstaplar tagna ifrån robotens tillhörande inbanor. Varje robot använder ett styrsystem som bygger på kundordrarna i Pågens affärssystem. Styrsystemet kontrollerar vilken brödsort och hur många lådor roboten skall hämta.

Figur 2.2 Översikt över de två modulbanorna samt några av inbanorna med robotar.

Om roboten får instruktioner att en stapel ska innehålla mer än en sorts bröd så gör roboten ett så kallat multiplock. Multiplock innebär att roboten först hämtar önskat antal brödlådor av den första sorten, sedan sätter roboten dessa brödlådor på en stapel av brödsort nummer två. Därefter gör roboten ett omtag där den även tar med sig önskat antal brödlådor från den andra sorten. När roboten hämtat alla efterfrågade brödsorter till stapeln, placeras den på en av modulbanorna. Det finns nu alltså en stapel med flera olika sorters bröd på modulbanan. Ett multiplock kan maximalt innehålla fyra olika sorters bröd eftersom roboten bara har fyra inbanor den kan hämta bröd ifrån. För att se till att alla staplar med brödlådor som ska till samma pall hamnar tillsammans finns det avsedda positioner på bandet. Varje position innehåller information om vilken lådstapel som skall placeras där. När en position närmar sig rätt plockrobot skickas ett meddelande med information om vad roboten skall hämta. När positionen kommer in i robotens arbetsområde lämnar den av lådstapeln på modulbanan den är avsedd för. Hinner inte roboten med så stoppas modulbanorna när positionen är vid slutet på robotens arbetsområde. På grund av att modulbanorna körs synkront så stoppas båda modulbanorna även om det bara är ena modulbanan som behöver stoppas.

Vid slutet av de två modulbanorna finns det fyra robotar som kallas för packrobotar. Deras uppgift är att lyfta ner brödstaplarna och ställa dem på pall. Robotarna arbetar i par och varje par arbetar med en pall åt gången. Eftersom brödstaplarna redan står i rätt ordning på modulbanorna så lyfter packrobotarna bara ner brödstaplarna och staplar de på pallen. Den information som packrobotarna håller reda på är, var på pallen brödstapeln ska stå och hur många staplar den ska ställa på varje plats. När en pall är full transporteras den iväg på rullbandet den står på och en ny tom pall åker fram till packrobotarna. Pallen som är full plastas in och transporteras vidare till utlastning där den sedan lastas på en lastbil och körs till kunden.

2.2 Simuleringsprogram

FlexSim är ett program som används för att simulera olika scenarion. Eftersom det ofta kan vara svårt att fatta komplicerade beslut är simulering ett kostnadseffektivt sätt till att testa olika idéer och teorier. En simuleringsmodell kan bland annat användas under designfasen av ett projekt, till exempel en produktionsanläggning, för att testa olika lösningar och hitta den bästa. Desto längre i arbetet pågått, desto mer ökar kostnaderna för att ändra på något. Anledningen till detta är att en ändring får större följdeffekter desto längre projektet pågått. Genom simulering går det att tidigt i arbetet fatta rätt beslut då olika teorier går att simulera och testa utan att behöva konstruera saker i verkligheten. Allt sker med hjälp av en dator och ett simuleringsprogram. FlexSim kan idag användas till att simulera saker inom sjukvården, gruvindustrin, tillverkning och materialhantering. För att simulera scenarion i dessa olika områden finns olika moduler av FlexSim som är speciellt anpassade för de olika områdena.

FlexSim använder sig av diskret händelsesimulering (Diskrete Event Simulation, DES). DES innebär att man modellerar händelserna i ett system som diskreta händelser. Varje händelse har sin specifika tidpunkt som den sker på och ändrar tillståndet på systemet, till skillnad från kontinuerlig händelsesimulering där händelserna sker kontinuerligt. I kontinuerlig händelsesimulering analyseras svaret från systemet kontinuerligt och sätter ekvationer som vanligen är differentialekvationer. I en kontinuerlig händelsesimulering finns alltså begränsade möjligheter till att snabbspola tiden då det krävs att allt i systemet dynamiskt ändras hela tiden. Med DES antas att inget i systemet ändras mellan två efterföljande händelser vilket ger möjligheten till att snabbt kunna hoppa fram i tiden och därmed snabbspola simulationen [3].

3. METOD

Detta kapitlet presenterar översiktligt projektets arbetsgång.

3.1 Orderplockanläggningen

För att skapa en förståelse för orderplockanläggningen som skulle simuleras användes en detaljerad 3D-modell gjord i programmet Google SketchUp. 3D-modellen tillsammans med videoklipp från delar av orderplockprocessen och handledning av projektledaren för orderplockprojektet gav insikt i processen från bageriet till färdiga pallar med bröd och gav bakgrunden till det som redovisats i kapitel 2.

3.2 Förstudie i FlexSim

Då inga förkunskaper om simulationsprogram fanns vid start av projektet, utgjordes de första veckorna av en självupplärningsfas. Målet med denna självupplärningsfas var att förstå de olika objekten i FlexSim, se kapitel 4.1 och hur flöden kan skapas med hjälp av dem. Materialet som lade grunden till förstudien var textboken ”Applied Simulation: Modeling and Analysis using FlexSim” [4] och hjälpkapitlen i FlexSim [5]. Med denna kunskap analyserades den befintliga simuleringsmodellen, gjord i version 5. Varje objekts kod undersöktes och kopplingarna mellan objekten följdes för att få förståelse för modellens uppbyggnad och funktion.

3.3 Migreringen

Migreringen planerades att i första hand göras genom att öppna senaste versionen av FlexSim och sedan öppna den befintliga simuleringsmodellen, för att därefter göra eventuella nödvändiga justeringar för att modellen ska fungera. Om det inte är möjligt att öppna den befintliga simuleringsmodellen i den nya versionen av FlexSim byggs en ny simuleringsmodell upp i den senaste versionen med hjälp av den befintliga simuleringsmodellen. Det första alternativet, alternativ 1, ger mest tillförlitligt resultat då den nya simuleringsmodellen kommer fungera som den befintliga. Detta på grund av att inga nya konstruktioner kommer behöva genomföras utan endast problemlösning av eventuella fel. Alternativ 2, att bygga upp en ny simuleringsmodellen från grunden, kommer ge ett mindre tillförlitligt resultat om uppbyggandet inte byggs precis efter den befintliga simuleringsmodellen. Att bygga precis som den befintliga simuleringsmodellen är inte garanterat att det går, då det skett ett antal uppgraderingar av FlexSim och funktioner har tagits bort och lagts till.

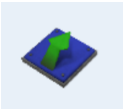
3.3 Modellverifiering

Den befintliga simuleringsmodellen som är gjord i version 5 skapades för att kunna beräkna kapaciteten för orderplockanläggningen innan den var byggd. Simuleringsmodellen utvärderas aldrig så osäkerheten var stor angående hur den fungerar jämfört med i verkligheten. För att kunna verifiera modellens funktionalitet mot verkligheten togs ny produktionsdata ifrån Pågen. Nyligen körda ordrar matades in i simuleringsmodellen och simuleringsstatistiken analyserades med statistik från Pågen.

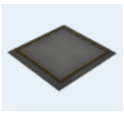
4. FLEXSIM

Simuleringar i FlexSim byggs upp med tre viktiga grundstenar. Den första grundstenen är objekten som utför uppgifter i simuleringen, sammankopplade med varandra och med flödesobjekt som passerar dem. Andra grundstenen är så kallade Labels som innehåller specifik information om objekt och flödesobjekt. Sista grundstenen är så kallade Triggers som styr över vad som händer i simuleringen. För att kunna bygga simuleringar krävs en grundlig förståelse hur de tre grundstenarna fungerar.

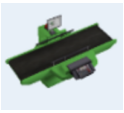
4.1 Objekt



Source – Producerar flödesobjekt i olika former. Kan bland annat producera efter ett visst tidsintervall, olika statistiska fördelningar, ett schema eller en viss sekvens. Skickar flödesobjektet till specifik port efter inställda villkor.

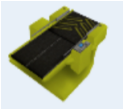


Queue – Förvara flödesobjekt i en kö med inställbar storlek. Kan både skicka och ta emot flödesobjekt från portar efter inställda villkor.



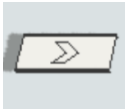
Processor – Simulerar en bearbetning med inställbar ställtid och cykeltid. Kan både skicka och ta emot flödesobjekt från portar efter inställda villkor.

Sink – Tar bort flödesobjekt ifrån modellen. Kan tex. användas istället för att bygga lager vid simulationsmodellens slut. Tar emot flödesobjekt från specifik port efter inställda villkor.



Combiner – Sammansätter, packar eller grupperar flödesobjekt med inställbar ställtid och cykeltid. Tar alltid först ett objekt från port 1 för att bygga på från de andra portarna tex. en lastpall och flera lådor. Skickar till specifik port efter inställda villkor.

Seperator – Separerar flödesobjekt genom att packa upp från t.ex. en pall eller gör en kopia av flödesobjektet efter inställda villkor. Har en inställbar ställtid och cykeltid. Tar emot och skickar flödesobjekt till portar efter inställda villkor.

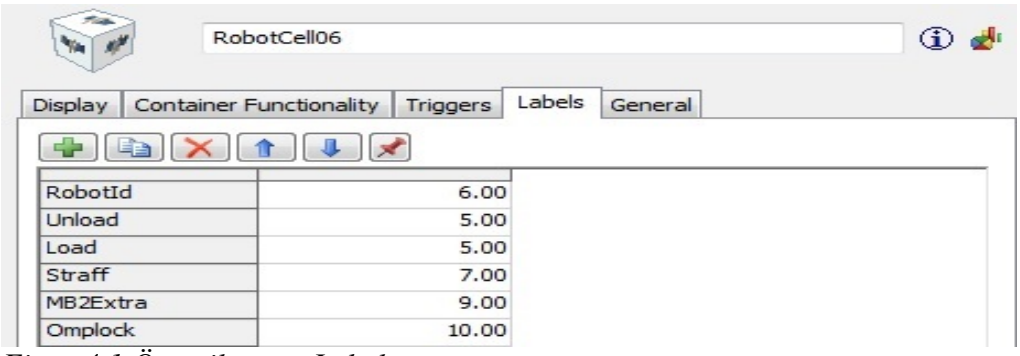


Conveyor – Simulerar transportband och kan byggas på med logik. Med en beslutspunkt kan logik byggas vid en viss punkt på transportbandet med vad som ska hända när ett flödesobjekt kommer. Fotocellen fungerar liknande men är alltid aktiv om ett flödesobjekt ligger i vägen.

Robot – Flyttar flödesobjekt från en startposition till en slutposition. Förflyttningstid, pålastningstid och avlastningstid styrs av inställbara villkor. Är inställbar i vilket rörelsemönster den ska åka, hur mycket den kan lyfta, vilken hastighet och acceleration den skall röra sig med.

4.2 Labels

Label är en slags etikett som är fäst på ett objekt eller flödesobjekt och innehåller information i textform eller nummerform. En Label kan både läsas från och skrivas till som för robotcellen i figur 4.1 nedan.

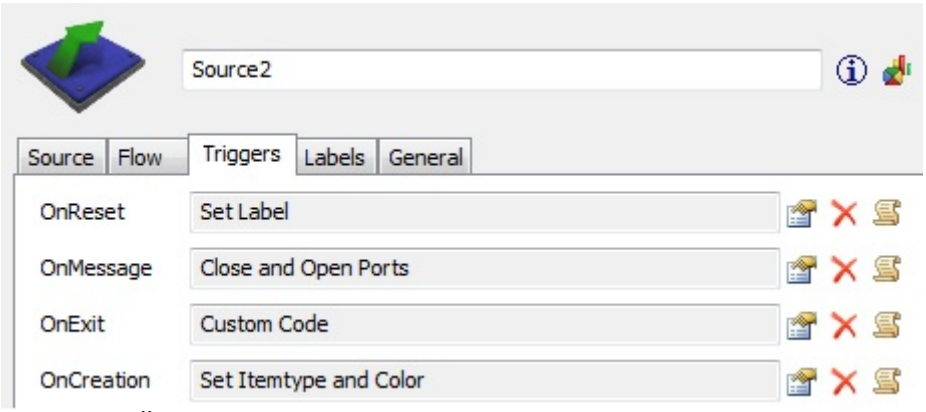


Figur 4.1 Översikt över Labels.

Flödesobjekt använder t.ex. Labels som RobotId för att kunna identifiera robotcellen och jämföra om det är denna robotcell den skall till. Robotcellen använder sina egna Labels för att kontrollera sina cykeltider och har också möjligheten att ändra flödesobjektets Labels efter avslutad operation.

4.3 Triggers

Mycket av den allmänna logiken för simulationsflöden i FlexSim byggs med triggers. Alla objekt har en triggersflik där det finns olika tidpunkter för det specifika objektet, se figur 4.2. Vid de olika tidpunkterna kan val göras mellan förinställda händelser som då utförs vid den valda tidpunkten. Det finns även möjlighet att skriva egen kod för att kunna utföra mer komplexa uppgifter. Programmeringsspråket som då används är FlexSims egna språk, FlexScript. FlexScript är väldigt likt #C men byggs på färdigskrivna funktioner som lämpar sig för simulationslogik.



Figur 4.2 Översikt Triggers.

Triggerslogiken har olika användningsområden baserat på tidpunkter. Triggern OnCreation kan t.ex. användas till ställa in färg och Labels på flödesobjekten som skapas.

4.4 Exempel på simuleringsscenario

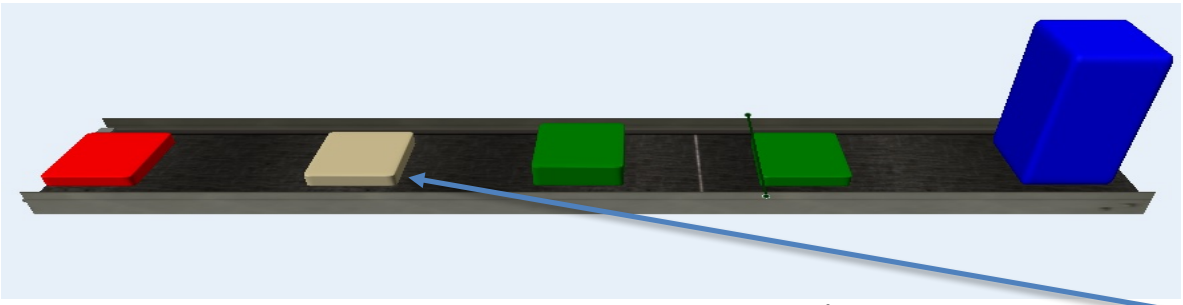
Ett enkelt simuleringsscenario i FlexSim kan t.ex. vara att simulera ett kösystem på en flygplats. Det finns två diskar där kunderna kan checka in sitt bagage och en kö. Beroende på om kunderna har en elektronisk biljett (E-biljett) eller pappersbiljett så tar det olika lång tid för dem att checka in sitt bagage vid disken. För att simulera hur kön bildas och hur många kunder i timmen systemet klarar av att hantera så kan man skapa två Sourcer, en Conveyor som simulerar kön, två Processorer som simulerar diskar och en Sink som tar bort flödesobjekten från modellen. I Sourcerna kan man då använda en Trigg som heter OnCreation för att t.ex. ange en viss färg på flödesobjektet beroende på om det är E-biljett eller pappersbiljett. Man kan samtidigt ange en Label till flödesobjektet men information om det är E-biljett eller pappersbiljett. Processorerna kan sedan läsa av Labeln för att ställa om processtiden beroende på vilken typ av kund det är. När kunden (flödesobjektet) sedan är färdigprocessad så tas det ur modellen. Man kan sedan lägga till att FlexSim för statistik över de saker som är intressanta för en själv och till exempel presentera detta i ett stapeldiagram. Detta är en väldigt enkel modell som går att göra mycket mer komplicerad med att lägga till fler objekt, villkor och använda fler Triggers och Labels.

5. BEFINTLIG MODELL

Den befintliga simuleringsmodellen är en modell som används till att simulera orderplockanläggningen. Den är dock gjord i FlexSim version 5 och ska lyftas till version 2016. Migrering ska ske genom att FlexSim konverterar modellen eller genom att modellen byggs upp från grunden om FlexSims konverteringen inte fungerar. Då det är från den befintliga simuleringsmodellen som arbetet kommer utgå ifrån kommer det behövas en grundlig förståelse för den befintliga simuleringsmodellen.

5.1 Inmatningen

På modulbanorna i modellen finns det flödesobjekt i form av gråa boxar för att symboliserar tomma platser där en robot kan ställa en stapel med brödlådor. När en plats fylls ändras storleken på boxen samtidigt som den färgsätts för att visualisera att en stapel med brödlådor ställts på modulbanan, se figur 5.1.



Figur 5.1 Modulbana som demonstrerar en tom plats som är grå & fyra stycken fyllda.

Skapandet av dessa platser sker av två source-objekt som är kopplade till en modulbana var. Source-objektet läser av en ordertabell avsedd för dess specifika modulbana och skapar platser med information från tabellen, se figur 5.2. Platsen som skapas för med sig all information från tabellen genom att använda Labels. De två modulbanorna i modellen är uppdelade i olika sektioner för att kunna tillhöra till de olika robotcellerna. På varje del av modulbanan får de plats max fem stycken platser och platserna har förbestämda avstånd mellan sig.

Order2010 Rows 1459 Columns 10 Use Bundle Clear on Reset Add Table to MTEI

	ArrivalTime	Namn	PallId	Quantity	RobotId	Antal	RobotBana	ModulBana	StapelId	StapelAntal
Row 1	0.00	134.00	1123155.00	1.00	6.00	8.00	3.00	1.00	1.00	2.00
	0.00	134.00	1123155.00	1.00	6.00	2.00	3.00	1.00	1.00	2.00
	0.00	134.00	1123155.00	1.00	7.00	8.00	3.00	1.00	2.00	2.00
	0.00	193.00	1123155.00	1.00	4.00	2.00	4.00	1.00	2.00	2.00
	0.00	134.00	1123155.00	1.00	6.00	8.00	3.00	1.00	3.00	3.00
	0.00	183.00	1123155.00	1.00	8.00	2.00	1.00	1.00	3.00	3.00

Figur 5.2 Tabell med indata för platserna på modulbana 1.

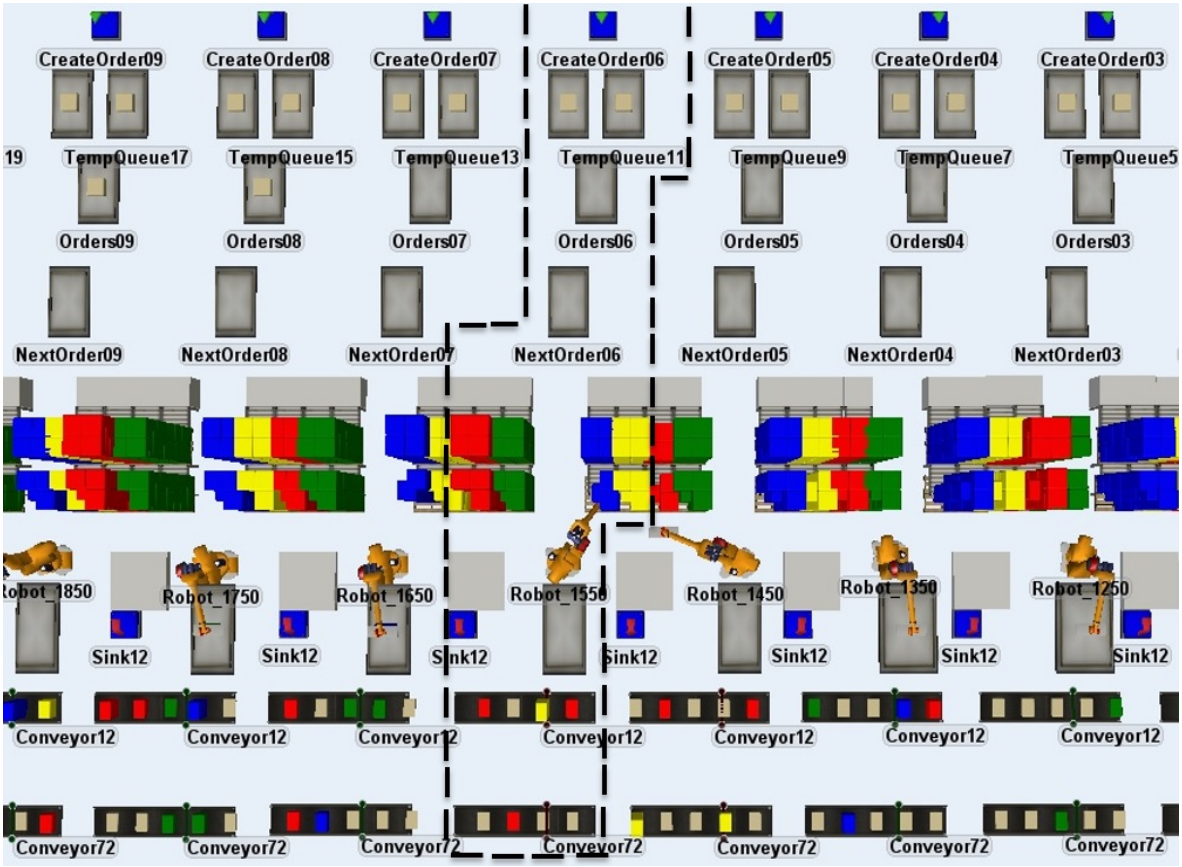
Platserna tar med sig orderinformationen genom följande Labels:

- **Klar** – Visar om en robot har placerat en stapel på platsen.
- **RobotId** – Anger vilken robot som ska hämta en stapel till platsen.
- **Antal** – Anger antalet brödlådor i stapeln.
- **RobotBana** – Anger vilken robotbana som roboten ska hämta brödstapeln ifrån.

- **ModulBana** – Anger vilken modulbana platsen finns på.
- **StapelId** – Anger vilken plats på pallen som stapeln ska placeras på. Det finns totalt fyra platser.
- **StapelAntal** – Anger hur många staplar som skall placeras på den specifika platsen på pallen.
- **PallIndex** – Anger vilken pall stapeln skall till.
- **ArtNr** – Anger vilken brödsort som finns i stapeln.

5.2 Robotcell

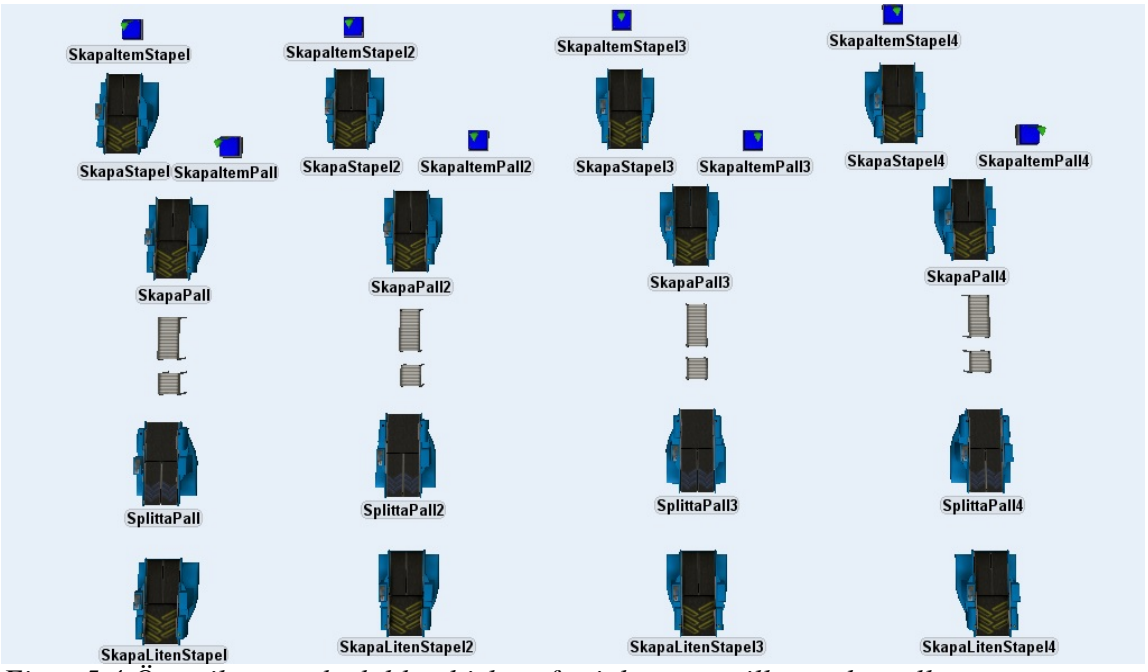
Modellen är indelad i tio robotceller med två tillhörande modulbanor för varje robotcell, se figur 5.3. När en ledig plats kommer in på modulbanan tillhörande en viss robotcell utförs en kontroll med hjälp av en triggersfunktion. RobotId för platsen jämförs med nästkommande robotcell för att se om roboten måste förbereda sig för att lämna av en brödstapel på platsen. Stämmer platsens och robotcellens RobotId överens så kopieras platsens Labels. Informationen skickas till första boxen i en kö som är kopplad till robotcellen. Kön kallas för ”TempQueueX” (där X är ett tal från 1-20), där varje robotcell har en av dessa köer för varje modulbana. När boxen tagit emot informationen skickas den vidare till den inbana den ska hämtas ifrån.



Figur 5.3 Översikt över robotceller.

Det finns fyra inbanor till varje robotcell med en specifik färg på brödstaplarna per bana. För varje inbana finns det ett antal objekt som är dolda, se figur 5.4. För att göra modellen mer lättöverskådlig och mer lik verkligheten.

De dolda objekten används både för att få till det visuella med staplar på robotinbanorna och för att ha koll på om roboten behöver extra tid vi hämtning. Ett flödesschema för att illustrera hur objekten är ihopkopplade finns i bilaga 1.



Figur 5.4 Översikt över de dolda objekten för inbanorna till en robotcell.

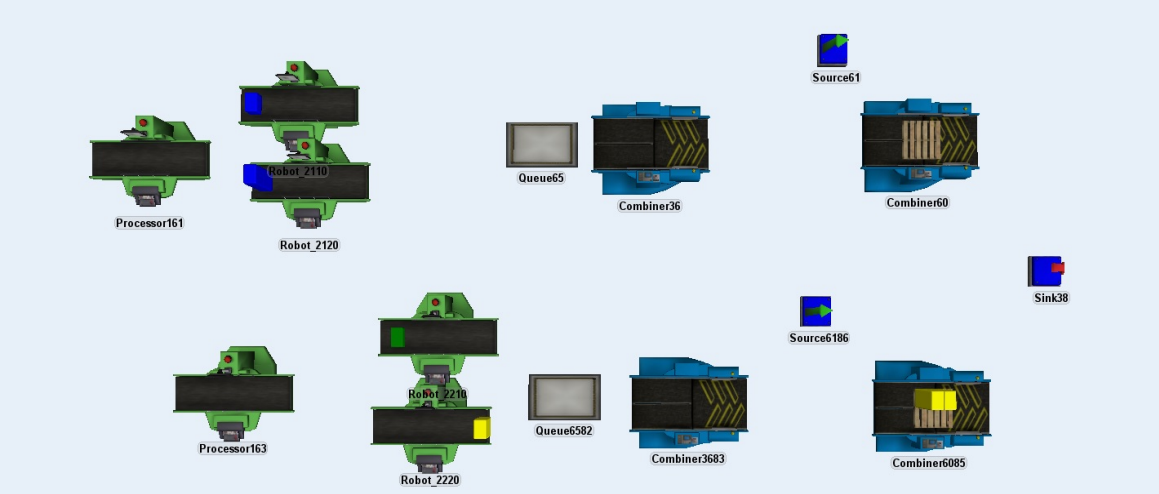
Från en tabell läser Sourcen SkapaItemStapel av hur många brödlådor det ska vara per pall på dess specifika inbana. Avläst antal boxar tillverkas i samma färg och skapas som en stapel av Combinern SkapaStapel. Dessa staplar läggs på en pall av Combinern SkapaPall. Pallarna skickas vidare på robotinbanorna och utgör de visuella brödpallarna som syns innan roboten.

Boxen som skapades med information ifrån den lediga platsen hamnar på den dolda Combinern SkapaLitenStapel på inbanan den skickades till. Boxen har via sina Labels kontroll på hur många brödlådor som roboten skall hämta. För att tillgodose detta tas en pall med brödstaplar från robotinbanan och separerar bort pallen med seperaton SplittaPall. Antalet efterfrågade brödstaplar skickas till Combinern SkapaLitenStapel som skapar ett objekt av brödstaplarna och boxen med boxens Labels. Skulle antalet brödstaplar i SplittaPall vara färre än efterfrågat tas en ny pall med brödstaplas ifrån robotinbanan. För att kompensera för tiden det tar för roboten att plocka ifrån en ny pall läggs då en strafftid på robotens förflyttningstid. Roboten plockar brödstapeln ifrån SkapaLitenStapel och lägger den i en kö.

I kön väntar stapeln på att platsen den ska till, kommer till modulbanan i den robotcellen. När platsen kommit fram lämnar stapeln kön och skickar information till platsen som ändrar färg och storlek för att visuellt ge en indikation på antalet brödlådor i stapeln och även vilken brödsort som finns in stapeln. Stapeln som lämnar kön åker till en Sink och lämnar modellen. Om roboten inte hinner med att lämna brödstaplarna i kön så stannar modulbanorna vid en fotocell som är vid slutet på robotens arbetsområde på modulbanan. När roboten har satt på en stapel startar modulbanorna igen.

5.3 Utmatningen

När en brödstapel kommer till slutet av modulbanorna vid den sista robotcellen används två uppsättningar med Processorer och Combineners, se figur 5.5. Dessa simulerar fyra robotar som plockar ner brödstaplar och ställer de på pall. Varje modulbana har en egen uppsättning som består av en Processor längst fram som alltid har processtiden noll förutom när Labeln pall-Id får ett nytt värde. Detta innebär att en ny pall måste matas fram vilket tar extra tid som då den första processorn då lägger till. De efterföljande Processor paret simulerar varsin robot vars rörelsetid är Processorns processtid. De efterföljande Combinerserna bildar pallar med brödstaplar och har koll på vart på pallen brödstaplarna skall stå. När pallen är klar tars den bort med en Sink som då registrerar att en pall har packats.



Figur 5.5 Översikt över hela utmatningen från modulbanorna.

6. MIGRATION FRÅN VERSION 5 TILL VERSION 2016

Här presenteras genomförandet, problem och övrig information om migrationen från FlexSim version 5 till FlexSim version 2016 av simuleringsmodellen.

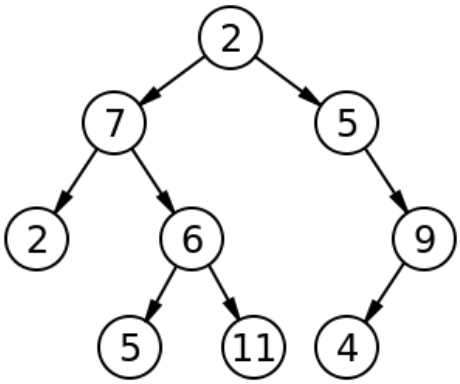
6.1 Uppdateringen

För att den befintliga simuleringsmodellen även ska vara användbar i framtida projekt valdes det att göra en migrering från version 5 till den nyaste versionen, FlexSim 2016. Migreringen ansågs främst kunna bidra till en simuleringsmodell med bättre grafik och funktionalitet. Förhoppningar fanns också för att ett nyare användargränssnitt skulle underlätta vid arbetet med modellen. Den befintliga simuleringsmodellen som var byggd i en sex år gammal version ansågs vara komplicerad att uppdatera eftersom mycket har hänt i utvecklingen av programvaran under dessa år. Vid projektets början fanns det två hypoteser att jobba efter. Att den gamla modellen skulle kunna öppnas i den nya versionen alternativt att modellen skulle behövas byggas upp från grunden. Förarbetet inför migrationsdelen av projektet var oberoende av vilken hypotes som skulle stämma. Detta på grund av att det krävdes samma förståelse för FlexSim vid båda utfallen.

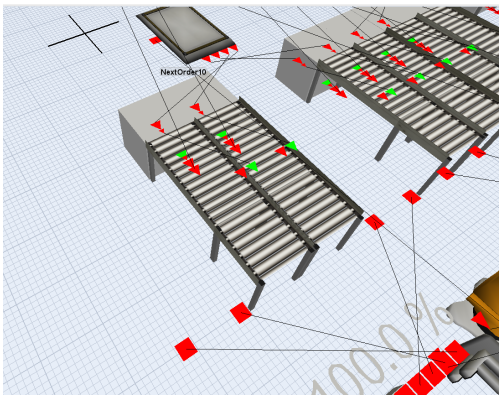
När den befintliga simuleringsmodellen öppnades i den nya versionen uppstod det inga felmeddelanden men det gick inte heller att visuellt se någon modell. Efter en dialog med FlexSims support testades det att öppna modellen stegvis i alla följande versioner från version 5 upp till version 2016. Detta resulterade i att modellen kunde öppnas korrekt och köras i några sekunder innan den automatiskt stoppades. Felmeddelandet som kunde utläsas klagade på att en variabel av typen int användes. Efter grundlig felsökning visade sig att version 5 använder sig av ett 32-bitars system och version 2016 använder sig av 64-bitars system. Kodningen är därför anpassad efter ett 32-bitar system och inte efter 64-bitars system. Problemet i koden uppstår för att variabeltypen int bara kan hantera 32-bitar vilket resulterade i att FlexSim gav ett felmeddelande om detta.

6.2 Variabeltyper

För att undkomma problemet med att versionerna använder sig av olika system valdes det att byta ut int-variabeln till en variabel av typen double. Int är en förkortning av ordet integer som betyder heltal och double står för double-precision floating-point-type. I ett programspråk som FlexScript så kan en integer hanterar heltal som består av 32-bitar, medan en double kan hantera reella tal och även mycket större tal som 64-bitars tal [6]. Det finns ytterligare en variabeltyp i modellen som förekommer ofta, och kallas treenode. Treenode används när man skapar variabler som ska referera till objekt i modellen. Alla objekten i modellen är strukturerade som ett träd där varje objekt eller nod som det också kallas har ”föräldrar” och ”barn”. I figur 6.1 kan man se att noden 6 har två barn, 5 och 11, och föräldern till noden 6 är 7 [7]. Ett exempel från modellen där ett objekt har flera barn återfinns på inbanorna till robotarna. Dessa inbanor ser visuellt ut som en grå fyrkant som kan ses i figur 6.2 där sedan några inbanor finns direkt efter. Denna fyrkant innehåller i sin tur nio stycken barn där ett av dom är Conveyorn som syns i modellen. Resten av barnen är dolda och finns till för att skapa pallarna med staplar som man sedan ser på inbanan.



Figur 6.1 Ett enkelt träd.



Figur 6.2 Inbanor till robotarna.

6.3 Metod för felsökning

Under projektets gång utvecklades en metod för att felsöka modellen. Det som nästan uteslutande alltid händer när det uppstår ett fel är att modellen stannar någonstans i en körning. För att hitta var problemet uppstår så började det att genomsöka vilken av de tio stycken robotcellerna eller de två packrobotcellerna som orsakade stoppet i modellen. När rätt robotcell var identifierad avgränsades felsökningsområdet till robotcellen, för att leta efter det objekt som orsakar stoppet i modellen. När rätt objekt som orsakar stoppet var identifierat applicerades så kallade breakpoints på lämpliga ställen, på den tillhörande koden till objektet. Breakpointsen gör att simuleringen stannar på raden där de finns, i avvaktan på att användaren stegar fram till nästa rad i koden eller återupptar simuleringen. Detta gjordes för att dra nytta av debug-funktionen som finns i FlexSim. Då objekt skickar meddelande mellan varandra uppstod ibland scenariot att andra objekt också undersöktes, på grund av att meddelanden skickades. Dessa objekt kunde antingen ligga inom det avgränsade felsökningsområdet eller utanför. Om nödvändigt sattes fler breakpoints på andra objekts kod för att systematiskt kunna stega sig igenom intressanta delar av koden i modellen. Löpande under felsökningsprocessen sker det nya avgränsningar av felsökningsområdet för att till slut kunna ringa in var felet uppstår.

6.4 Implementering av ändringar

Efter att ha bytt ut alla variabler av typen int till double som kan hantera 64-bitar uppstod det ett helt annat fel. Det gick varken att implementera ändringar eller att spara modellen. Det nya felmeddelandet talade om att licensen till FlexSim som användes har en begränsning på max 100 objekt i en och samma modell. Eftersom modellen har fler än 100 objekt resulterade detta till att det inte gick att applicera några ändringar. Eftersom Teamster själva inte hade någon FlexSim licens av FlexSim 2016 hade det tidigare beslutats att använda en studentversion av FlexSim. Kontakt med en återförsäljare av FlexSim hade bekräftat att en studentversion ska fungerar att arbeta i då studentversionen av FlexSim inte har några begränsningar. Den enda skillnaden mellan studentversionen och den fullvärdiga versionen av FlexSim är att studentversionen har en vattenstämpel tvärs över skärmen. Men efter att stött på problemet med att versionen inte kunde hantera modeller med 100 objekt eller fler kontaktades återförsäljaren igen. Efter ett par dagar kom ett svar från återförsäljaren att han gjort ett misstag och tillhandahållit fel version av FlexSim samt att detta skulle åtgärdas. Därefter tillhandahölls en ny licens utan

begränsningen på 100 objekt och det gick nu att implementera förändringarna på modellen.

Med rätt licens till FlexSim gick det nu att implementera ändringarna. Dock upptäcktes ganska snart att modellen stannade på precis samma ställe som innan trots att ändringarna av variabeltypen implementerats. En ledig plats som är ämnad för en viss robotcell stannade framför robotcellen i väntan på en brödstapel. Problemet var att ingen brödstapel hämtades till den lediga platsen på modulbanan vilket medförde att systemet stod stilla i väntan på händelsen. Vid närmare undersökning av vad som blev fel i robotcellen upptäcktes det att boxen med all information till brödstaplarna satt fast i köerna innan inbanorna. Köernas portar som släpper in och ut objekt styrs med meddelandetriggers. Med hjälp av metoden för felsökning och debug funktionen i FlexSim upptäcktes det att detta meddelande aldrig skickades ut. Det visade sig att de olika variabeltyperna ställde till problem här också. En treenode returnerades nu till en double och som då fick ett felaktigt värde. För att få bort problemet byttes noden till en double. Slutligen valdes det dock att byta ut alla double till typen treenode, vilket är ett mer korrekt alternativ kodmässigt på grund av att variabeln var en referens till det flödesobjekt som skulle släppas från kön. Efter dessa ändringar kunde modellen köras helt från start till färdigpackade pallar.

6.5 Jämförelse

För att kunna säkerställa att uppdateringen och ändringarna i modellen inte haft påverkan på modellens simuleringsresultat gjordes jämförelse med tidigare körningar. I ett Exceldokument finns tidigare körningars resultat antecknade med information om modell och indata. Då dokumentet går in på väldigt detaljerad statistik valdes det att fokusera jämförelsen på kapacitet med parametrarna körtid, antal pall, antal staplar och antal pall per timme. Dessa parametrar stämde exakt överens med den nya körningen. Därför ansågs uppdateringen vara slutförd och projektet gick vidare till nästa fas.

Jämfört med FlexSim version 5 har användargränssnittet på FlexSim uppdaterats till ett modernare utseende i version 2016. Grafiken på simuleringsmodellen har förbättrats samt många nya funktioner har lagts till. Saker som förbättrats är till exempel Conveyorsen och en funktion som nu gör det möjligt att sampla referenser när man skriver egen kod har lagts till.

7. Verifiering av modell

Simuleringsmodellen skapades i samband med orderplockprojektet för att kunna beräkna kapaciteten av orderplockanläggningen. Det har dock aldrig skett någon total verifiering på hur bra modellen överensstämmer med orderplockanläggningen i Malmö. Genom att köra samma parti i den uppdaterade modellen som nyligen är körd i orderplocksanläggning kan de bådas statistik jämföras.

7.1 Inmatning av kördata

Orderplockanläggningen i Malmö arbetar efter ordrar från Pågens affärssystem. Ordrarna innehåller all ny data in till anläggningen som behövs för att skapa färdiga pallar med bröd till kund. I figur 7.1 syns ett utdrag från en tabell med all data. Varje rad i tabellen är en intern order till en viss robotcell som sedan lyfter ut en brödstapel på modulbanan. Den interna ordern har information om till vilken pall brödstapeln skall till och vilken robotcell som ska hantera ordern. I tabellen så finns det två viktiga återkommande namn på kolumn F till O, ArtNr och Antal och med 10,20,30 eller 40 på slutet. Siffrorna på slutet anger vilken inbana roboten ska hämta brödstaplarna ifrån. ArtNr innehåller information om vilken brödsort som ska hanteras och Antal säger hur många brödlådor som ska hämtas. De två gulmarkerade raderna är interna ordrar med mer än en sorts bröd. Detta innebär att roboten behöver gör ett multiplock för att ha flera sorters bröd i samma stapel.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	IDNr	Indx	PlockRbtNr	StapelBana	ArtNr10	Antal10	AntalPåPall10	ArtNr20	Antal20	AntalPåPall20	ArtNr30	Antal30	AntalPåPall30	ArtNr40	Antal40	AntalPåPall40
2	9881859	1123155	6	1	0	0	0	0	0	0	134	8	8	0	0	0
3	9881860	1123155	6	1	0	0	0	0	0	0	134	2	2	0	0	0
4	9881861	1123155	7	1	0	0	0	0	0	0	134	8	8	0	0	0
5	9881862	1123155	4	1	0	0	0	0	0	0	0	0	0	193	2	2
6	9881863	1123155	6	1	0	0	0	0	0	0	134	8	8	0	0	0
7	9881864	1123155	8	1	183	2	2	0	0	0	0	0	0	0	0	0
8	9881865	1123155	7	1	0	0	0	0	0	0	134	1	1	0	0	0
9	9881866	1123155	4	1	0	0	0	0	0	0	0	0	0	193	10	10
10	9881879	1123156	8	1	0	0	0	0	0	0	0	0	0	352	10	10
11	9881880	1123156	8	1	0	0	0	0	0	0	0	0	0	352	3	3
12	9881881	1123156	9	1	0	0	0	0	0	0	0	0	0	352	10	10
13	9881882	1123156	6	1	0	0	0	353	2	2	0	0	0	0	0	0
14	9881883	1123156	3	1	347	1	1	0	0	0	0	0	0	0	0	0
15	9881884	1123156	10	1	0	0	0	0	0	0	355	10	10	0	0	0
16	9881885	1123156	7	1	0	0	0	0	0	0	0	0	0	352	3	3
17	9881886	1123156	8	1	0	0	0	0	0	0	0	0	0	352	4	4
18	9881887	1123156	10	1	0	0	0	131	2	2	355	7	7	0	0	0
19	9881867	1123157	10	2	0	0	0	0	0	0	0	0	0	314	10	10
20	9881868	1123157	3	2	0	0	0	118	3	3	0	0	0	0	0	0
21	9881869	1123157	2	2	0	0	0	0	0	0	118	8	8	0	0	0
22	9881870	1123157	1	2	0	0	0	311	1	1	0	0	0	0	0	0
23	9881871	1123157	10	2	0	0	0	0	0	0	0	0	0	314	1	1
24	9881872	1123157	2	2	0	0	0	0	0	0	118	3	3	340	2	2
25	9881873	1123157	3	2	0	0	0	0	0	0	119	4	4	0	0	0
26	9881874	1123157	10	2	0	0	0	0	0	0	0	0	0	314	1	1

Figur 7.1 Excel-fil från Pågens ordersystem.

FlexSimmodellen använder likt orderplockanläggningen interna ordrar i tabellformat för att kunna skapa färdiga pallar med bröd. De tabeller som följde med modellen är lika gamla som modellen och är från början tagna från riktiga körningar. För att tabellerna från Pågen skall kunna fungera i modellen behöver de ha en annan ordning. Att ändra detta manuellt via klipp och klistra funktioner i Excell ansågs som den smidigaste lösningen vid få körningar. Vid flera byten av indatan till modellen blir detta dock väldigt tidskrävande. För att göra modellen mer användarvänlig beslutades det att gör det möjligt att i FlexSim kunna mata in och köra med original tabellerna med ordrar via enkla knapptryckningar.

FlexSim har en inbyggd funktion där det kan importeras in tabeller ifrån Excell. Problemet är att FlexSimmodellen läser av informationen ifrån tabeller i en annan ordning än vad originaltabellen står i. Dessutom används en tabell för vardera modulbana. Två lösningar på detta problem utvärderades närmare. Antingen anpassas all kod i modellen till de ordertabellerna som är direkt tagna från Pågen eller så görs dessa tabeller om till den form modellen läser ifrån. Att anpassa all kod efter tabellen skulle bli ett väldigt svårt arbete då modellen är väldigt komplex och har kod i många olika objekt. Mycket av koden skulle också behövas skrivas om helt. Att skriva kod som gör nya tabeller ansågs som det bästa och säkraste sättet. Detta på grund av att det bidrog till mindre ändringar i modellen och gick snabbare att koda.

Rows 1362 Columns 20 Use Bundle Clear on Reset Add Table to MTEI Add Table to MTEE

PallId	ArtNr1	Antal1	BanNr1	ArtNr2	Antal2	BanNr2	ArtNr3	Antal3	BanNr3	ArtNr4	Antal4	BanNr4	RobotID	StapelId	StapelAntal
1123155.00	134.00	8.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	6.00	1.00	2.00
1123155.00	134.00	2.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	6.00	1.00	2.00
1123155.00	134.00	8.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.00	2.00	2.00
1123155.00	193.00	2.00	4.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.00	2.00	2.00
1123155.00	134.00	8.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	6.00	3.00	3.00
1123155.00	183.00	2.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	8.00	3.00	3.00
1123155.00	134.00	1.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.00	3.00	3.00
1123155.00	193.00	10.00	4.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	4.00	4.00	1.00
1123156.00	352.00	10.00	4.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	8.00	1.00	2.00
1123156.00	352.00	3.00	4.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	8.00	1.00	2.00
1123156.00	352.00	10.00	4.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	9.00	2.00	3.00
1123156.00	353.00	2.00	2.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	6.00	2.00	3.00
1123156.00	347.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	3.00	2.00	3.00
1123156.00	355.00	10.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	10.00	3.00	2.00
1123156.00	352.00	3.00	4.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.00	3.00	2.00
1123156.00	352.00	4.00	4.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	8.00	4.00	2.00
1123156.00	131.00	2.00	2.00	355.00	7.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	10.00	4.00	2.00
1123159.00	134.00	8.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	6.00	1.00	2.00
1123159.00	134.00	3.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.00	1.00	2.00
1123159.00	134.00	8.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	9.00	2.00	2.00
1123159.00	134.00	3.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.00	2.00	2.00
1123159.00	134.00	7.00	3.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.00	3.00	2.00
1123159.00	131.00	4.00	2.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	10.00	3.00	2.00
1123159.00	131.00	7.00	2.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	10.00	4.00	3.00
1123159.00	133.00	1.00	1.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	7.00	4.00	3.00

Figur 7.2 Inmatad tabell i FlexSim som är omgjord av programkod.

När tabellen har gjorts om med programkoden i FlexSim ser den ut som i figur 7.2 De två tabellerna som skapas är Oder2010 för modulbana 1 och Order2020 för modulbana 2. Programkoden sorterar först ut ordrarna efter vilka modulbanor de skall till. Sedan består koden till mestadels av att ändra ordning på kolumnerna så att det stämmer överens. Det är dock inte enbart att flytta alla kolumnerna till rätt ordning utan några parametrar som behövs i modellen måste räknas fram.

Ett exempel på detta är tabellen från Pågen där det finns kolumner med 10, 20, 30 och 40-ändelser. Här representerar ändelserna i kolumnerna robotens 4 inbanor. Om en brödstapel skall hämtas från inbana 3 står artikelnumret i kolumnen ArtNr30. I FlexSimmodellen används istället kolumnerna ArtNr med ändelserna 1, 2, 3 och 4. Dessa representerar de antal olika brödstaplar roboten skall hämta i en intern order. Om roboten bara skall hämta en sorts bröd så står artikelnumret på kolumn ArtNr1. För att veta vilken av robotens inbanor den ska hämta ifrån finns kolumnen BanNr med samma ändelse som ArtNr. Koden som möjliggör detta och all annan omvandling kan hittas i bilaga 2.

7.2 Multiplock

Vid projektets start fanns flera simuleringsmodeller att tillgå. Vid studerande av de olika simuleringsmodellerna sågs det visuellt inga skillnader mellan modellerna. Därför valdes den simuleringsmodellen som var skapad senast. Simuleringsmodellen som valdes har senare i projektet visat sig att den inte har funktionalitet för multiplock, att kunna hämta flera olika brödstaplar samtidigt. Vid efterforskning visade det sig att det fanns annan modell som hade stöd för detta. Detta resulterade i att allt som hade byggts upp och utvecklats i den andra modellen också behövde implementeras på den nya simuleringsmodellen med multiplocksfunktionen. Med den nya multiplocksfunktionen krävdes det också att inmatningen av ordertabellerna behövde modifieras.

Multiplocksfunktionen i den nya modellen kontrollerar om en ledig plats är avsedd till mer än en sorts bröd. Är den det så sätts en Label för att indikera detta till roboten. När roboten sedan ska plocka de olika lådorna sätt en extra gångtid eftersom roboten behöver göra ett eller flera omtag.

7.3 Simulerad produktionsstatistik

Statistikdelen i simuleringsmodellen har modifierats så att statistiken skrivs ut från FlexSim på den formen och i det dokumentet som de gamla körningarna är dokumenterade i. Statistiken kan nu enkelt exporteras genom en knapptryckning. Anledningen till att det valts att presentera statistiken på samma sätt är för att det lätt ska gå att jämföra körningar mellan den befintliga modellen och den nya.

7.4 Verifiering mot Pågenstatistik

För att kunna verifiera modellen mot den fysiska anläggningen på Pågens bageri i Malmö behövdes både indata och utdata från produktionen i orderplockanläggningen hos Pågen. Det loggas statistik över alla körningar på orderplockanläggningen tio dagar bakåt i tiden. Med detta som utgångspunkt valdes fyra nattkörningar, 22-23, 23-24, 24-25 och 25-26 Maj 2016, till att jämföra modellen med verkligheten. Då loggfilerna från körningarna består av alla meddelande som anläggningen rapporterat i följd, var det tvunget att filtrera bort information som inte var väsentlig. Efter studerande av loggfilerna hittades meddelanden med meddelandekod 251 som angav att en pall blivit klar. Dessa meddelande filtrerades fram för att kunna studera klockslag och pall-id för att sedan kunna bestämma vilka exakta tider och pallar på körningarna på som skulle jämföras. När starttid och sluttid samt pall-id på första och sista pallen för fyra olika körningar var framplockade, kunde indata och körstatistik plockas fram. Körstatistiken presenterades dock inte på samma sätt som i statistiken från modellen. Det skiljde även mellan vad som presenteras, till exempel finns nyttjandegrad av robotarna i statistiken från modellen men detta finns inte i statistiken från verkligheten. Detta medförde att statistiken fick tolkas och matchas mot varandra för att kunna jämföra modellen med verkligheten. Körstatistiken från båda modellerna sammanställdes sedan i ett Excel-dokument för att enkelt kunna jämföra dessa två. En procentuell felskillnad mellan modellen och den verkliga anläggningen räknades också ut enligt ekvation 7.1 för att enkelt kunna se hur mycket modellen avviker från verkligheten.

$$Procentuell\ skillnad = \frac{Modell - Verklig\ anläggning}{Verklig\ anläggning} \quad (7.1)$$

Den sammanställda jämförelsen visar att modellen simulerar verkligheten bra. I parametern pall/h som syns i figur 7.3 så visar modellen en medelvärdesavvikelse på 0,17% av de fyra utvalda körningarna. Varför just pall/h parametern är intressant är för att den på ett bra sätt visar anläggningens kapacitet. Vid början av projektet var man osäker vilken kapacitet anläggningen skulle klara av. Som en del av förstudien till att ta reda på detta gjordes simuleringsmodellen i FlexSim. Detta är en av de mest intressanta parametrarna att få så bra som möjligt då modellens huvuduppgift är att ge en korrekt kapacitet på orderplockanläggningen. Parametrarna Pall, Staplar, Stapel/pall, Pall/h, Stapel/h och körtid beskriver kapaciteten på anläggningen. Kapacitetsparametrarna är på de fyra körningarna som valts ut visar aldrig mer än 5%, vilket får anses som ett okej resultat med tanke på den begränsade tiden för projektet. Dessa parametrar syns alla i tabell 7.1.

De parametrar som har störst fel är de som är kopplade till tillgängligheten på modulbanorna. Med tillgänglighet på modulbanorna menas hur ofta modulbanorna behöver stoppas förutom det scenariot att modulbanorna stoppas pga. att robot inte hinner med. Dessa parametrar är även de som varit svårast att jämföra i statistiken med verkliga modellen då de parametrar som tillsammans bildar tillgängligheten inte kunnat matchas på ett helt korrekt sätt mellan modellen och verkligheten. Till exempel har en helt korrekt synkronisering av modulbanorna i modellen inte lyckats implementeras. Detta kan ses i statistiken från modellen där den loggade gångtiden på modulbanorna inte är samma mellan modulbanorna. Gångtiden borde vara densamma då det är tänkt att modulbanorna ska gå synkront. Dock är synkroniseringen skapad genom att modulbanorna tvingas att producera lika många staplar vilket i slutändan gör att modulbanorna i det stora hela har gått synkront med varandra. Eftersom modellen är ett helt system som simuleras hänger allting ihop på ett eller annat sätt. Om tillgängligheten kan bättras på så kommer högst troligen att parametern Pall/h också förbättras då de hänger ihop och vice versa.

	22-23	23-24	24-25	25-26	Genomsnitt
Antal pallar	1,99%	5,08%	-3,41%	-2,46%	0,30%
Antal staplar	1,29%	4,57%	-2,48%	-1,82%	0,39%
Staplar/pall	-0,72%	-0,46%	0,91%	0,64%	0,09%
Pall/h	1,45%	5,14%	-3,35%	-2,57%	0,17%
Staplar/h	0,83%	4,56%	-2,47%	-1,81%	0,28%
Körtid	0,00%	0,00%	0,00%	0,00%	0,00%
Total tillgänglighet	21,03%	25,85%	24,51%	23,07%	23,61%
MB1 tillgänglighet	23,86%	26,86%	28,41%	25,53%	26,17%
MB2 tillgänglighet	18,43%	24,83%	20,54%	20,68%	21,12%
MB1 körtid	3,85%	2,80%	-3,75%	8,08%	2,75%
MB2 körtid	0,20%	0,33%	-8,21%	7,18%	-0,12%
MB1 antal staplar	-1,80%	3,48%	-4,77%	-2,85%	-1,49%
MB2 antal staplar	4,59%	5,67%	-0,07%	-0,76%	2,36%
MB1 går	1,97%	1,48%	-4,85%	5,97%	1,14%
MB2 går	-0,77%	-0,98%	-9,42%	5,17%	-1,50%

Tabell 7.1 Visar avvikelsen mellan modellen och verkligheten i procent.

8. Resultat

Den befintliga simuleringsmodellen har inte behövts byggas upp från grunden då det visade sig vara möjligt att migrera modellen genom FlexSim. För att kunna migrera modellen krävdes det att modellen öppnades i alla versioner mellan versionen 5 till version 2016. Efter att modellen kunde öppnas i version 2016 var den dock inte fungerande. Ett stort arbete med felsökning krävdes innan modellen kunde ses som uppdaterad med samma funktionalitet som den gamla modellen. Då modellen aldrig behövdes byggas upp från grunden underlättade inte modellen någon uppbyggnad. Att det fanns en bra simuleringsgrund i denna modell var gynnsamt för att undgå att behöva skapa en så komplex simulationsmodell helt från grunden. Resultatet av uppgraderingen är att modellen har blivit mycket bättre grafiskt och att det möjliggör framtida påbyggnader och justeringar till liknande projekt.

En av fördelarna med att uppgraderingen till den nyaste versionen av FlexSim är att användargränssnittet har blivit väldigt bra. Den stora skillnaden grafiskt har gjort det mer inspirerande att arbeta i programmet. De nya funktionerna har bidragit till smidigare programmering av simuleringen. En funktion som speciellt har varit till stor nytta är möjligheten att sampla en referens med ett klick. I de tidigare versionerna behövdes ett objektets hela namn skrivas ut i programmeringskoden. Nu räcker det att klicka på samplingsverktyget och sedan på det specifika objektet. De nya funktionerna har också påverkat de olika simulationsobjekten som Conveyors. Dessa objekt har blivit smartare men flyttbara beslutspunkter där användaren kan fritt placera ut vart en trigger ska bli påverkad. Eftersom de gamla Conveyorna följde med i migreringen ansågs det vara ett för stort jobb att ersätta alla dessa med de nya då de innehåller mycket kod.

Allt arbete som har lagts ner i projektet på olika håll har gjorts för att slutligen kunna göra en grundlig verifiering utav modellens riktighet gentemot verkligheten. Fyra olika dagars körningar från Pågen gav en stabil jämförelse över hur statistikparametrarna låg. Motsvarande tid kördes med samma ordertabell i FlexSim som skrev ut alla statistikparametrar till ett Excel-dokument. Varje körnings parametrar jämfördes med statistiken från Pågenkörningarna. De viktigaste parametrarna så som hur mycket pallar anläggningen kan framställa skiljde sig bara ett fåtal procent. Den enda mätparametern som skiljde sig mycket från verkligheten var tillgängligheten. Det var också denna parameter som var svår att jämföra mellan simuleringsstatistiken och den verkliga. Tillgängligheten bygger på en del av stoppstatistiken, den presenterades med olika parametrar i simuleringen och verkligheten. Möjliga förbättringsåtgärder på modellen för att den ska bli ännu mer överensstämmande är att undersöka de olika stoppen i modellen. Statistiken från Pågen har fler parametrar för stopp än FlexSim. Tillgängligheten för FlexSimmodellen räknas ut med larm för packrobotarna och plockrobotarna. Pågenstatistiken räknar även med en viss del av parametern stopp på grund av synkronkörning vid tillgängligheten. Detta ligger ihop klumpad för stopp av packrobotar och plockrobotar i statistiken från FlexSimmodellen. Därför skiljer tillgängligheten så mycket mellan verkligheten och simuleringen. För att kunna jämföras helt med verkligheten så borde det finnas en parameter för stopp på grund av synkrondrift i simuleringen. Just synkrondriften har stor förbättringspotential för att få modellen mer exakt då modulbanorna inte kör exakt synkront.

9. Diskussion

Här ges personliga reflektioner om resultaten och projektet i sin helhet.

9.1 Om FlexSim

Då tidigare erfarenheter av simuleringsprogram helt saknats har en stor del av projektet fått läggas på att få förståelse för simuleringar i FlexSim. Det förarbetet som främst har tagit tid är arbetet med att förstå hur den befintliga modellen är uppbyggd. Detta arbete har indirekt varit helt avgörande för projektets resultat. Då modellen är väldigt komplex och stor har de mest avancerade funktioner i FlexSim behövts att användas. Detta har gjort att det hela tiden krävts väldigt grundlig förståelse för programmet. Tidigare programmeringserfarenheter från kurser på Chalmers har varit gynnsamma för kunna förstå logiken i FlexSims programmeringsspråk Flexscript. Problemen har brutits ner i delar för att lättare kunna hanteras. Detta har gjort det enklare att kunna applicera de kunskaper utbildningen bidragit med för att kunna lösa problemen.

Eftersom modellen skapades år 2012 med stor hjälp av FlexSims support fanns det ingen kunskap om FlexSim och modellen längre inom Teamster. Detta medförde att all information fick arbetas fram självständigt via att studera modellen och läsa i FlexSims hjälpkapitel. Denna okunskap om modellen har bidragit till mycket arbete som skulle kunnat undvikas som arbete med en modell helt utan multiplockfunktion. Det har också uppkommit många funderaringar om varför saker är gjorde på vissa sätt och om detta verkligen är den bästa lösningen. Dessa frågor har då inte kunnat bli besvarade så avvägningar har alltid behövts göra ifall nya lösningar ska implementeras. Ett exempel på detta är skapandet av platser på modulbanorna. I modellen med multiplocksfunktion gjordes detta av endast en Source. I koden såg det ut som Sourcen skickade varannan order till modulbana 1 och varannan till modulbana 2. Detta oavsett vilken modulbana ordern använder i körningen i Pågens orderplocksanläggning. Då det inte fanns någon att fråga varför den agerade annorlunda gentemot verklighet beslutades det att implementera en annan lösning. Istället användes en Source till varje modulbana för att efterlikna verkligheten.

9.2 Om migreringen

I början av projektet fanns det två hypoteser. Den första hypotesen var att det skulle gå att uppdatera den befintliga modellen till den senaste versionen genom att öppna modellen i den senaste versionen av FlexSim. Hypotes nummer två utgick ifrån att den första hypotesen inte skulle vara möjlig. Därför skulle hela simuleringsmodellen behöva byggas upp från grunden. Vid planering av projektet utgicks det från hypotes nummer två eftersom den ansågs mest omfattande och tidskrävande. Vid migreringen visade det sig var den första hypotesen som stämde. Då inte simuleringen i den nya versionen fungerade behövdes inte examensarbetets omfattning att ändras. Istället upptogs tiden med felsökning för att få igång simuleringen i den komplexa modellen.

Eftersom det har tagit mycket tid att felsöka modellen kan det utvärderas om det hade gått snabbare att bygga upp en modell från grunden istället. Fördelen med att bygga upp modellen från grunden är att kodningen skulle kunna ske efter eget programmeringstänk.

Då skulle förståelse funnits för varför det varit programmerat på ett visst vis. Detta skulle medföra att felsökningen skulle bli mycket lättare.

Nu när modellen har studerats har det funnits objekt som det inte har gått att hitta någon förklaring till vad de gör för nytta eller varför de finns. Med tanke på modellens komplexitet med dess anpassning efter FlexSims begränsningar så anses ändå uppbyggnaden från grunden var mer tidskrävande och svårare.

9.3 Om verifieringen

Verifieringen visade att FlexSimmodellen bara skiljde några få procent från den verkliga orderplockanläggningen i de allra flesta mätparametrarna. Parametern som skiljde mest var tillgängligheten. Tillgängligheten i modellens statistik räknades ut från larmstopp på packrobotarna och plockrobotarna. I statistiken från Pågen tas även en viss del av parametern ”stopp på grund av synkronkörning” med i tillgänglighetsuträkningen. Ska tillgängligheten stämma bättre överens krävs det att den räknas ut på samma sätt.

Från orderplocksanläggning så går det bara att se vilken tid en viss pall blir klar. När statistik tas ut från den tiden så är modulbanan redan full med brödstaplar och går. På grund av detta tar det inte lång tid tills de nästkommande pallarna blir klara. Vid körning av samma pallar i FlexSim börjar modulbanorna tomma. Detta gör att det tar ett par minuter innan första pall blir klar. Detta är en felorsak i verifieringen, dock påverkar detta väldigt lite i en körning på många timmar.

9.4 Om vidareutveckling av modellen

Projektet har nått upp till alla på förhand uppsatta mål. Modellen skulle dock utanför examensarbetets ramar kunna utvecklas vidare. Det som kan utvecklas är att göra den ännu mer överensstämmande med verkligheten. När modellen förbättrats för att få mer korrekt simulering av verkligheten kommer den även kunna användas till att optimera den verkliga anläggningen. Genom att simulera modellen kan man övervaka intressanta parametrar för att upptäcka så kallade flaskhalsar i anläggningen. När flaskhalsarna identifierats kan dessa sedan åtgärdas för att få en anläggning med högre kapacitet. För att åtgärda flaskhalsarna kan simuleringsmodellen också användas. Den kan användas till att testa olika lösningar på åtgärder för att ta fram den optimala lösningen.

Referenser

1. Pågen [internet]. Wikipedia; 2016 [Hämtad: 2016-06-07]. Tillgänglig från: http://pagen.se/om-pagen/press/

2. 14 robotar och 1700 pallar bröd per dag [internet]. Nicklas Dahlin; 2010 [Senast ändrad: 2010-08-18; Hämtad: 2016-06-07]. Tillgänglig från: http://www.nyteknik.se/automation/14-robotar-och-1700-pallar-brod-per-dag-6424835

3. Introduction to Discrete-Event Simulation and the SimPy Language [internet]. Norm Matloff; 2008 [Hämtad: 2016-06-07] Tillgänglig från: http://heather.cs.ucdavis.edu/~matloff/156/PLN/DESimIntro.pdf

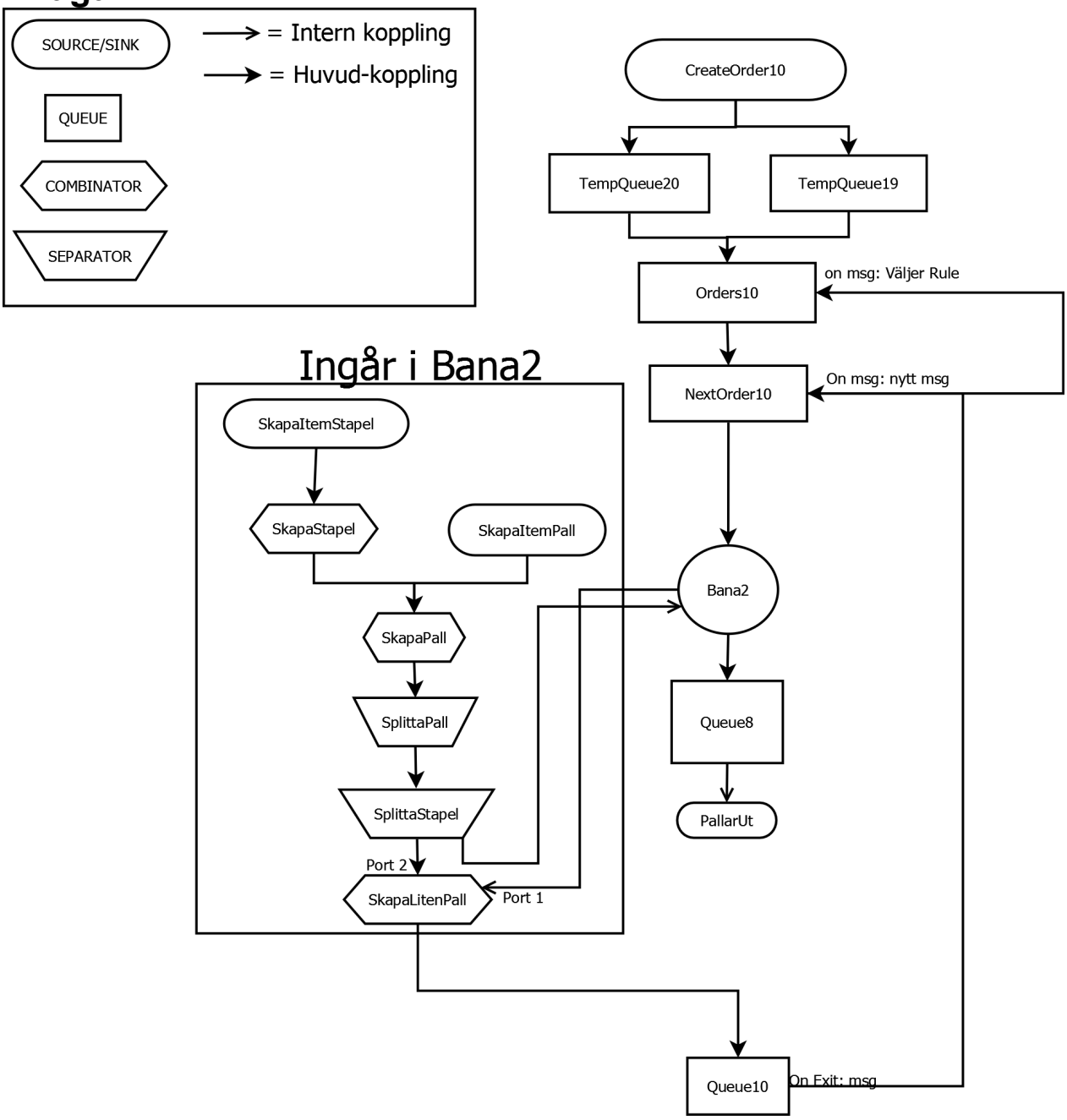
4. Beaverstock M, Greenwood A, Nordgren W. Applied Simulation: Modeling and Analysis using FlexSim [internet]. Upplaga 4. Orem, Utah: FlexSim Software Products, Inc. Canyon Park Technology Center; 2011 [Hämtad: 2016-06-07]. Tillgänglig från: https://www.flexsim.com/store/

5. User Manual: FlexSim 2016 [internet]. 2016-03-14 [Hämtad: 2016-06-07]. Tillgänglig från: https://www.flexsim.com/account/#/downloads

6. C Data types [internet]. ARMmbed [Hämtad: 2016-06-07]. Tillgänglig från: https://developer.mbed.org/handbook/C-Data-Types

7. Tree [internet]. NIST(National Institute of Standards and Technology); 2008 [Senast ändrad: 2008-08-14; Hämtad: 2016-06-07]. Tillgänglig från: https://xlinux.nist.gov/dads//HTML/tree.html

Bilaga 1


SOURCE/SINK
= Intern koppling
= Huvud-koppling
QUEUE
COMBINATOR
SEPARATOR
CreateOrder10
TempQueue20
TempQueue19
Orders10
on msg: Väljer Rule
NextOrder10
On msg: nytt msg
Ingår i Bana2
SkapaItemStapel
SkapaStapel
SkapaItemPall
SkapaPall
SplittaPall
SplittaStapel
Port 2
SkapaLitenPall
Port 1
Bana2
Queue8
PallarUt
Queue10
On Exit: msg

Bilaga 2

```
/**Custom Code*/
treenode current = ownerobject(c);
int numOfRows2010 = 0;
int numOfRows2020 = 0;

numOfRows2010 = gettablerows("Order2010");
numOfRows2020 = gettablerows("Order2020");

setlabelnum(current,"Antal",1);

//Tar bort alla rader förutom första och nollställer denna. Gäller tabellerna
Order2010 och Order2020
for(int i = numOfRows2010; i > 1; i--){
      deletetablerow("Order2010",i);
}
clearglobaltable("Order2010");

for(int i = numOfRows2020; i > 1; i--){
      deletetablerow("Order2020",i);
}
clearglobaltable("Order2020");

//**************************************
int rad = 1;
int column = 0;
int n = 0;
intarray tempAntal = makearray(4);
intarray tempBana = makearray(4);
intarray tempArtNr = makearray(4);

numOfRows2010 = 1;
numOfRows2020 = 1;

for(rad = 1; rad <= gettablerows("Order"); rad++){
      for(int i = 1; i <= 4; i++){
            tempArtNr[i] = 0;
            tempBana[i] = 0;
            tempAntal[i] = 0;
      }

      if(gettablenum("Order",rad,4) == 1){
            if(numOfRows2010 > 1){
                  addtablerow("Order2010");
            }
            settablenum("Order2010",numOfRows2010,2,gettablenum("Order",rad,2));
            //Sätter PallId i Order2010


      settablenum("Order2010",numOfRows2010,15,gettablenum("Order",rad,3));
      //Sätter RobotID i Order2010

      settablenum("Order2010",numOfRows2010,16,gettablenum("Order",rad,18));
      //Sätter StapelId i Order2010
```

```
settablenum("Order2010",numOfRows2010,20,gettablenum("Order",rad,1));
//Sätter OrderId i Order2010

      //***************10/20/30/40***************************
      for(int i = 1; i <= 4; i++){
            switch(i){
                  case 1:
                        column = 6;
                        tempBana[i] = 1;
                        break;
                  case 2:
                        column = 9;
                        tempBana[i] = 2;
                        break;
                  case 3:
                        column = 12;
                        tempBana[i] = 3;
                        break;
                  case 4:
                        column = 15;
                        tempBana[i] = 4;
                        break;
            }
            if(gettablenum("Order",rad,column) != 0){
                  tempAntal[i] = gettablenum("Order",rad,column);
                  tempArtNr[i] = gettablenum("Order",rad,(column-1));
            }
      }
      int plats = 0;
      column = 3;
      n = 0;
      for(int i = 1; i <= 4; i++){
            if(tempAntal[i] != 0){
                  plats = n*3;
            settablenum("Order2010",numOfRows2010,column +
            plats,tempArtNr[i]);
            //Sätter ArtNr1/2/3/4 i Order2010

            settablenum("Order2010",numOfRows2010,column+1 +
            plats,tempAntal[i]);
            //Sätter Antal1/2/3/4 i Order2010

            settablenum("Order2010",numOfRows2010,column+2 +
            plats,tempBana[i]);
            //Sätter BanNr1/2/3/4 i Order2010

                  n++;
            }
      }
      //************************************************************
```

```
	//**********Sätter OrderAntal och AntalTot***************
	int antal = gettablenum("Order2010",numOfRows2010,4) +
	gettablenum("Order2010",numOfRows2010,7) +
	gettablenum("Order2010",numOfRows2010,10) +
	gettablenum("Order2010",numOfRows2010,13);

	settablenum("Order2010",numOfRows2010,19,antal);



	if (gettablenum("Order2010",numOfRows2010,12) != 0)
	{
		settablenum("Order2010",numOfRows2010,18,4);
	}
	else if(gettablenum("Order2010",numOfRows2010,9) != 0)
	{
		settablenum("Order2010",numOfRows2010,18,3);
	}
	else if(gettablenum("Order2010",numOfRows2010,6) != 0)
	{
		settablenum("Order2010",numOfRows2010,18,2);
	}
	else
	{
		settablenum("Order2010",numOfRows2010,18,1);
	}
	//*********************************************************


	numOfRows2010++;
}
else if(gettablenum("Order",rad,4) == 2){
	if(numOfRows2020 > 1){
		addtablerow("Order2020");
	}
	settablenum("Order2020",numOfRows2020,2,gettablenum("Order",rad,2));
	//Sätter PallId i Order2020


settablenum("Order2020",numOfRows2020,15,gettablenum("Order",rad,3));
//Sätter RobotID i Order2020

settablenum("Order2020",numOfRows2020,16,gettablenum("Order",rad,18));
//Sätter StapelId i Order2020


settablenum("Order2020",numOfRows2020,20,gettablenum("Order",rad,1));
//Sätter OrderId i Order2020
```

```
		//***************10/20/30/40***************************
		for(int i = 1; i <= 4; i++){
			switch(i){
				case 1:
					column = 6;
					tempBana[i] = 1;
					break;

				case 2:
					column = 9;
					tempBana[i] = 2;
					break;
				case 3:
					column = 12;
					tempBana[i] = 3;
					break;
				case 4:
					column = 15;
					tempBana[i] = 4;
					break;
			}
			if(gettablenum("Order",rad,column) != 0){
				tempAntal[i] = gettablenum("Order",rad,column);
				tempArtNr[i] = gettablenum("Order",rad,(column-1));
			}
		}
		int plats = 0;
		column = 3;
		n = 0;
		for(int i = 1; i <= 4; i++){
			if(tempAntal[i] != 0){
				plats = n*3;
				settablenum("Order2020",numOfRows2020,column +
plats,tempArtNr[i]);		//Sätter ArtNr1/2/3/4 i Order2020

				settablenum("Order2020",numOfRows2020,column+1 +
plats,tempAntal[i]);		//Sätter Antal1/2/3/4 i Order2020

				settablenum("Order2020",numOfRows2020,column+2 +
plats,tempBana[i]);		//Sätter BanNr1/2/3/4 i Order2020
				n++;
			}
		}
		//*********************************************************

		//**********Sätter OrderAntal och AntalTot***************
		int antal = gettablenum("Order2020",numOfRows2020,4) +
		gettablenum("Order2020",numOfRows2020,7) +
		gettablenum("Order2020",numOfRows2020,10) +
		gettablenum("Order2020",numOfRows2020,13);

		settablenum("Order2020",numOfRows2020,19,antal);
```

```
            if (gettablenum("Order2020",numOfRows2020,12) != 0)
            {
                  settablenum("Order2020",numOfRows2020,18,4);
            }
            else if(gettablenum("Order2020",numOfRows2020,9) != 0)
            {
                  settablenum("Order2020",numOfRows2020,18,3);
            }
            else if(gettablenum("Order2020",numOfRows2020,6) != 0)
            {
                  settablenum("Order2020",numOfRows2020,18,2);
            }
            else
            {
                  settablenum("Order2020",numOfRows2020,18,1);
            }
            //************************************************************

            numOfRows2020++;
      }
}
//--------------------------------Sätter StapelAntal (kolumn 10)---------------
int rad2010 = 1;
int rad2020 = 1;
int stapelID2010 = 0;
int stapelID2020 = 0;
int stapelAntal = 0;

stapelID2010 = gettablenum("Order2010",rad2010,16);

while(rad2010 < gettablerows("Order2010")){
      while(gettablenum("Order2010",rad2010,16) == stapelID2010){
            stapelAntal++;
            rad2010++;
      }
      rad2010 = rad2010 - stapelAntal;
      for(int i = 1; i <= stapelAntal; i++){
            settablenum("Order2010",rad2010,17,stapelAntal);
            rad2010++;
      }
      stapelAntal = 0;
      stapelID2010 = gettablenum("Order2010",rad2010,16);
}
stapelID2020 = gettablenum("Order2020",rad2020,16);

while(rad2020 < gettablerows("Order2020")){
      while(gettablenum("Order2020",rad2020,16) == stapelID2020){
            stapelAntal++;
            rad2020++;
      }
      rad2020 = rad2020 - stapelAntal;
      for(int i = 1; i <= stapelAntal; i++){
            settablenum("Order2020",rad2020,17,stapelAntal);
            rad2020++;
      }
      stapelAntal = 0;
      stapelID2020 = gettablenum("Order2020",rad2020,16);
```