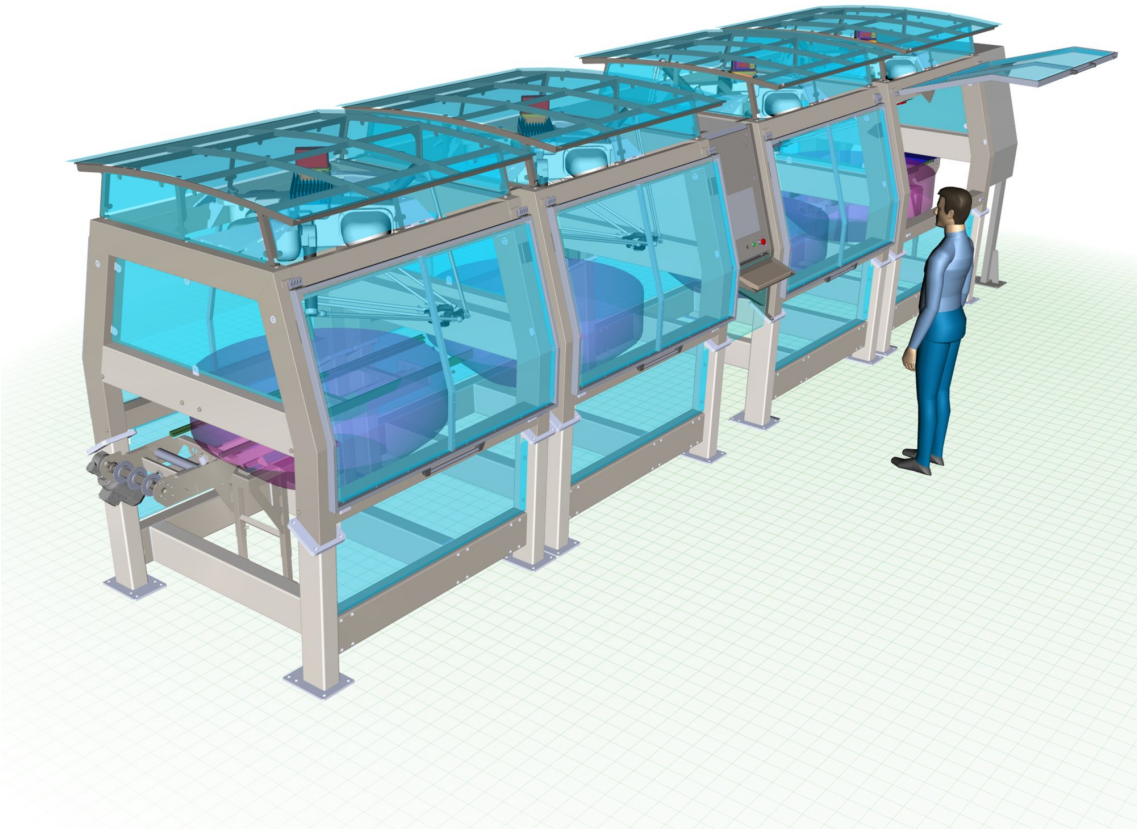
Udvikling af simulering af automatiseret pakkeanlæg



Aalborg Universitet
Institut for Mekanik og Produktion
Virksomhedsteknologi 4. semester

Jan Christian Conradsen
Søren Dahl Christensen

Institut for Mekanik og Produktion

Fibigerstræde 16,
DK-9220 Aalborg Ø
Phone: +45 9940 8934
Fax: +45 9815 3040
http://www.production.aau.dk

Titel:
Udvikling af simulering af automatiseret pakkeanlæg

Semester tema:
Virksomhedsteknologi

Projekt periode:
02-02-2011
- 31-05-2011

Projekt gruppe:
14.33a

Deltagere:
Søren Dahl Christensen
Jan Christian Conradsen

Vejleder:
Ole Madsen

Oplagstal: 5

Antal sider: 119 + Appendiks

Afsluttet: 31-05-2011

Synopsis:

I dette projekt blev der udarbejdet et brugervenligt og fleksibelt simuleringsværktøj, som kan simulere et pakkeanlæg bestående af 1-8 robotceller, 2-3 transportbånd (1 til emner, 1-2 til kasser) og er programmeret med C#. Værktøjet giver nøgletal for hvordan anlægget performer, og gør det muligt at udvikle og evaluere nye pakkestrategier.
Til at udvikle simuleringsværktøjet blev det undersøgt hvorledes software generelt udvikles og efterfølgende dokumenteres. Analysering viste at realtidsskedulering ikke var muligt. Derfor kan man med simuleringsværktøjet opbygge pakkestrategier udfra de tre regler *heuristiske prioriteringsregler*, *vertikal prioritering* og *minimum antal targets*.
En række forsøg blev udført på i alt 12 forskellige pakkestrategier og resultaterne af disse forsøg udmøntede i en pakkestrategi, P6, som anbefales at blive testet yderligere.
Yderligere anbefalinger kan læses i rapporten.

Abstract

This project was done in collaboration with OCTOMATION robotic workforce formerly known as InMóTx and was a continuation of an earlier project made on 9th semester. The overall purpose was to increase the performance of an automated packaging system. Today OCTOMATION find it difficult to test modifications on the packaging system and therefore they want to do it virtually. The benefit is less resources spent on testing and the possibility to reproduce these tests.

It was decided to programme a simulation tool in C#. Software developing was studied and gave some methods on how to develope software and how to define the specifications of the software. Furthermore it gave some design guidelines for developing an user interface and how to document the architecture.

Analysing was commenced in order to study how to increase the performance of an automated packaging system. The speed by which the parts and boxes arrives in the packaging system is very high. It was concluded that scheduling methods which calculate the optimal solution can not be calculated fast enough. Therefore some heuristic prioritisation rules have been studied. These prioritisation rules will not give the optimal solution, but they will give a satisfactory solution. Furthermore some others rules and constrains are examined. Together with the prioritisation rules it is possible to create packaging strategies.

The simulation tool has been produced. The design guidelines from the analysis has been used which make the user interface user-friendly. It is possible to setup packaging strategies from the three basic rules of *heuristic prioritsation rules*, *vertical prioritisation* and *minimum number of targets*. These packaging strategies can then be tested in various packaging systems and under different workload circumstances. When the simulation is running, it is possible to see key performance indicators and graphs. When the simulation is completed, it is possible to process and analyse the results in MATLAB. The simulation tool has been validated in two steps. First the individual components

have been validated to see if they operate as intended. Subsequently the entire simulation system were tested in order to validate that no errors would occur, when the components were put together. The simulation system performed with no errors and the validation was satisfactory.

The simulation tool was used to conduct a study on how various packaging strategies would perform under different circumstances. The results gave no precise answer to which strategy were best. However the combination called P1 were best at minimising the number of not-packed boxes. Furthermore it can be recommended to use FIFO in the first robot cell for boxes. The results from the test were used to create a packaging strategy called P6, which uses all the best elements from the testings. It is recommended that further testing are conducted with this strategy.

This project has all in all resulted in a user-friendly and flexible simulation tool which OCTOMATION can use to develop and test new packaging strategies without using an excessive amount of resources.

Forord

Denne rapport er udarbejdet i forbindelse med 4. semesters projektperiode på studieretningen Virksomhedsteknologi ved Aalborg Universitet. Det overordnede tema for projektet er "Virksomhedsteknologi".

Projektet er dokumenteret ved hovedrapport, appendiks samt bilags-cd. Hovedrapporten kan læses selvstændigt, men underbygges af appendiks og litteraturhenvisninger.

Hovedrapporten er for læsevenlighedens skyld delt op i kapitler og afsnit. Projektarbejdet har dog haft en iterativ karakter, selvom afsnittene i rapporten er forsøgt delt op.

Kildehenvisninger er opsat efter Chicago-metoden, hvilket i teksten er angivet som [x,år], hvor x er forfatterens efternavn og år er udgivelsesåret. Hvis der refereres til flere kilder skrives disse [x,år], [y,år]. Referencer til specifikke sidetal er angivet som [x,år,s. a-b]. Hvis der refereres til en forfatter, der har udgivet flere materialer samme år, angives det først anvendte materiale med et "a" efter årstallet. Materialer derefter angives b,c,d... alt efter rækkefølgen for dets optræden i rapporten.

I referencelisten er kilder generelt angivet på følgende måde:

Efternavn, fornavn. Titel. Forlag/URL. år.

Henvises der til materiale på bilags-cd'en er dette vist som: "filnavn.type" og eventuelt ark "navn".

Figurer, formler og tabeller er nummereret fortløbende, med kapitelnummer og herefter figurens nummer, eksempelvis "Figur 4.1". Appendiks angives på samme måde, blot med bogstaver som kapitelangivelse.

Bagerst i rapporten er der vedlagt en bilags-cd, som indeholder:

- Rapporten(PDF)
- PDF filer som er anvendt til litteraturhenvisning
- octoSimulation
- Test indstillinger og resultater
- MATLAB statistik modul

Tak til

Peter Nielsen, Aalborg Universitet

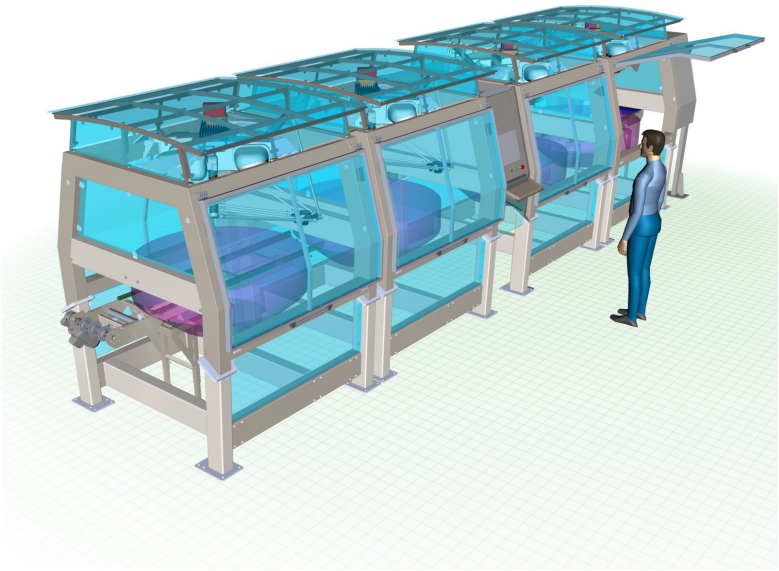
Jan Christian Conradsen

Søren Dahl Christensen

Indhold

1 Indledning

OCTOMATION er en virksomhed, tidligere kendt som InMóTx, som udvikler og sælger automatiserede pakkeanlæg til fødevareindustrien, hvor et sådan automatiseret pakkeanlæg kan ses på figur 1.1. De har et mål om at deres automatiserede pakkeanlæg skal være nogle af de mest brugervenlige, fleksible, robuste og effektive som findes på markedet [OCTOMATION, 2011]. Til at opnå dette, ønsker de at kunne afprøve nogle ændringer i pakkelæggenes styring, som sandsynligvis vil kunne øge ydelsen og gøre deres pakkeanlæg endnu mere effektive.



Figur 1.1: Her ses et automatiseret pakkeanlæg bestående af fire robotter, et visionsystem og tre transportbånd, som står hos virksomheden Rahbek fisk [RAHBEK, 2011] [OCTOMATION, 2011].

Dette projekt tager udgangspunkt i pakkeanlægget stående hos Rahbek fisk, som de anvender til at pakke frosne fiskeprodukter [RAHBEK, 2011]. Til at pakke disse frosne fisk, anvendes fire robotter, et visionsystem og tre transportbånd, hvor sammenspillet mellem disse enheder udføres i en styring. Pakkeanlæggets styring er opbygget således at udfra den samlede belastning på hele anlægget, reguleres hvorledes emnerne fordeles til de enkelte robotter, hvilket kaldes for Loadsharing. Denne Loadsharing-strategi mener OCTOMATION at der er mulighed for at forbedre, men det at afprøve ændringer i pakkeanlæggets styring giver nogle problemstillinger. Disse problemer består bl.a. af, at ændringer i pakkeanlæggenes styring kræver mange ressourcer i form af

mandetimer, når disse skal testes, og der er ingen mulighed for at reproducere testene og sammenligne resultatet af ændringerne. Derfor blev det i et for-projekt på 9. semester konkluderet, at en simulering af pakkeanlægget vil kunne afhjælpe disse problemstillinger. Dette medførte, at der blev undersøgt hvilke simuleringsværktøjer, der kunne anvendes til opbygge en simulering af deres pakkeanlæg. I tabel 1.1 ses de programmer som blev testet.

Program	**Leverandør**
DELMIA	Dassualt Systemes [Dassault Systemes, 2011]
RobWork	SDU [RobWork, 2011]
Experior	Xcelgo [Xcelgo A/S, 2011]
Adept V+ Emulator	Adept [Adept Technology Inc., 2011]
Enterprise Dynamics	InControl [InControl, 2011]
Eget program	Projekt gruppen

Tabel 1.1: Her ses de simuleringsprogrammer der blev testet i for-projektet.

Der blev opstillet nogle krav som de enkelte programmer blev testet op imod, hvilket er: Flow af emner og kasser, Procestider og Loadsharing-strategi. Disse krav ønskes opfyldt på følgende måder, som ses i det nedenstående og efterfølgende forklares.

- Flow af emner og kasser
  - Indbygget Input Source
  - Data input
- Procestider
  - Data input
  - Kinematisk og dynamisk model
- Loadsharing
  - Indbygget
  - Pakkeanlæggets styresystem (ACE)

Flow af emner og kasser er et nødvendigt krav, da der ønskes at teste og forbedre ydelsen på pakkeanlægget, og dermed kræver det at kunne simulere et flow af emner og kasser. Inputtet af disse skal enten komme fra en InputSource, som med en korrekt varians og fordeling skal bestemme flowet. Et andet muligt input er at indhente data fra Rahbek pakkeanlægget, og anvende disse direkte i simuleringen.

Procestider kan ligeledes komme fra data, som dog kræver at der meget præcist vides hvor emnet og target befandt sig for den påglædende procestid. En anden mulighed er at få indført en kinematisk og dynamisk model af robotten, hvilket vil gøre det muligt, at udføre robottens bevægelse fra punkt A til punkt B og bestemme procestiden.

Et andet krav er Loadsharing, som bliver anvendt i pakkeanlæggets styring til at styre hvorledes emner fordeles til robotcellerne. Denne del skal også kunne implementeres, som enten en indbygget del i simuleringen eller en løsning, hvor simuleringen tilkobles pakkeanlæggets styring og anvender Loadsharing-strategien herfra.

Der er i overensstemmelse med OCTOMATION besluttet, at projektgruppen selv udvikler en simulering i C#. Dette skyldes en tilfredshed med den beta-simulering, som blev programmeret på 9. semester. Denne simuleringen opfyldte delvis de tre stillede krav og giver derfor et godt grundlag til videreudvikling og udarbejdelse af en endelig simulering. OCTOMATION ser også en mulighed for at simuleringen kan programmeres til, at det bliver muligt at tilkoble deres eget statistik modul. Dette statistik modul er programmeret i C#, og bruges i pakkeanlæggets brugerflade, OCTOSOFT, til at vise ønsket statistik over belastning og ydelse af anlægget. En sådan tilkobling vil give en ønskværdig mulighed for sammenligning af statistik fra pakkeanlægget og simuleringen.

1.1 Initierende problem

OCTOMATION har et ønske om at øge fleksibiliteten ved at kunne håndtere variationen af indkomne emner, hvilket på nuværende tidspunkt ikke kan lade sig gøre. Dette medfører at arbejdsbelastningen fordeles ujævnt. Deres ønske er derfor at fordele belastningen mere jævnt mellem robotterne i pakkeanlægget. Yderligere er der et ønske om at pakkeanlæggets ydelse maksimeres, og derved er der nogle kriterier som skal opfyldes for at opnå dette.

Til at opnå dette ønskes det at minimere og maksimere følgende kriterier, som ses i tabel 1.2 på den følgende side.

Minimeres	Maksimeres
-	- Den homogene fordeling af arbejdsbelastningen
-	- Udnyttelsesgraden på pakkeanlægget
- Antal ubehandlet emner	- Antal behandlet emner
- Antal ikke færdig pakket kasser	- Antal pakket kasser

Tabel 1.2: Her ses de kriterier som ønskes minimeres og maksimeres.

Disse kriterier er ikke uafhænginge, og derfor er det svært at overskue konsekvenser ved at ændre nogle tiltag. Derfor bliver opgaven at få opbygget en simulering i C#, hvor det vil være muligt at afprøve nogle tiltag, som gerne skal kunne minimere og maksimere disse kriterier. Det skal derfor undersøges hvorledes sådanne kriterier kan minimeres/maksimeres. Yderligere ønskes det at anvende metoder til udvikling af software, som gerne skal hjælpe med at opnå en form for systematik i udviklingen af simuleringen samt efterfølgende, at kunne dokumentere denne på en overskuelig måde. Derfor kan følgende initierende problem opstilles:

> *Undersøg metoder som kan anvendes til at forbedre den måde pakkeanlægget pakker på, således at ydelsen forbedres samt hvordan generel udvikling af software udføres.*

Disse problemstillinger vil i det efterfølgende kapitel blive analyseret, hvilket skal give en forståelse for, hvordan en sådan problemstilling gribes an og hvordan software udvikles og dokumenteres.

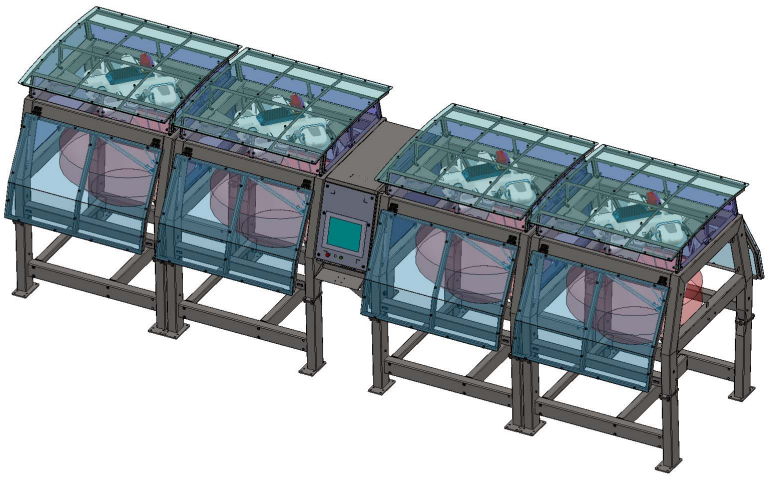
Analyse

2

I analysen vil det først blive undersøgt, hvordan dimensionerne for anlægget Rahbek er. Derefter undersøges teorien bag software udvikling, som baggrundsviden til udvikling og dokumentering af simuleringssystemet. Der vil til sidst i analysen blive undersøgt, hvordan en pakkestrategi skal sættes sammen, hvilke valg den skal indeholde, samt undersøge om realtidsskedulering er en reel mulighed.

2.1 Dimensioner

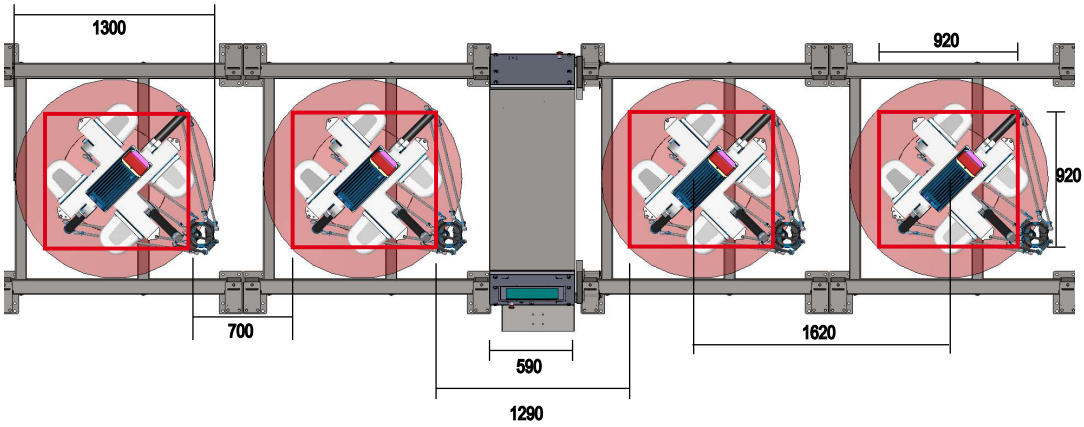
I dette afsnit bestemmes diverse mål der skal anvendes i simuleringen. Det har ikke været muligt at få en arbejdstegning af Rahbek pakkeanlægget, men i stedet er alle komponenterne modtaget som CAD tegninger og samlet i SolidWorks, som det ses på figur 2.1. Herefter er de ønskede mål bestemt.



Figur 2.1: Her ses pakkeanlægget bestående af fire robotceller og kontrolboksen i midten. Robotternes arbejdsrum er markeret som en rød cylinder samt en keglestub i hver robotcelle.

De mål som skal anvendes i simuleringen er størrelsen af robotternes arbejdsområde og afstande mellem disse. På figur 2.2 på den følgende side er målene indtegnet og det antages, at transportbåndenes højde gør at robotternes arbejdsområde bliver i cylinderområdet. Arbejdsområdet er indtegnet som firkant, hvilket antages at være den måde OCTOMATION har defineret deres arbejdsområde i pakkeanlæggets styring.

Hermed er mål til simuleringen bestemt:



Figur 2.2: Her ses pakkeanlægget set fra oven hvor de øverste komponenter er skjult, og størrelsen af robotternes arbejdsområder og afstande mellem disse er indtegnet.

Udvalgte mål	Mål[mm]
Maks. arbejdsområde	920x920
Længde mellem arbejdsområder	700
Bredde af controllerboks	590

Tabel 2.1: Her ses udvalgte mål, som skal bruges i simuleringen.

2.2 Software teori

I dette kapitel beskrives metoder til udvikling af software, som vil være anvendelig til at udvikle simuleringen. Der er en række krav til simuleringen som skal klarlægges inden udviklingen af simuleringen påbegyndes. Disse krav stilles både af kunden, OCTOMATION, og nogle system krav opstilles af udvikleren, hvilket er vigtige for at det ønskede stykke software fungere som tiltænkt. Yderligere vil der beskrives hvordan en brugervenlig brugerflade opbygges samt hvordan software dokumenteres, så det bliver overskueligt for andre end udvikleren selv.

2.2.1 Procesmodeller

Software udvikles for det meste efter nogle software procesmodeller, som alle indeholder en række processer der skal udarbejdes. Forskellen mellem disse procesmodeller er måden hvor på de processer udføres. De to meste anvendte procesmodeller er vandfaldsmodel og iterationsmodel [Sommerville, 2007, s. 9]. De processer som udar-

bejdes i procesmodellerne er:

1. Specifikation - I denne proces klarlægges specifikationer til det ønskede software mellem kunden og udvikleren
2. Udvikling - I denne proces designes og programmeres det ønskede software
3. Validering - I denne proces valideres og verificeres softwaren
4. Videreudviklig - Denne proces anvendes kun, hvis der efter levering er brug for en modificeret udgave af det eksisterende stykke software, således det tilpasses og opfylder kundens nye behov

Specifikation af softwaret er en proces, hvor der defineres hvilke krav og ønsker kunden stiller. Det er vigtigt for udvikleren at få klarlagt og beskrevet disse krav samt fortælle kunden, hvilke muligheder og begrænsninger der er mht. udviklingen af softwaren. Denne fase er meget kritisk, da misforståelser her vil medføre problemer senere i de andre processer. Dette vil især være i valideringsprocessen, hvor kunden skal godkende softwaret. Specifikationerne bliver normalt opstillet på to niveauer, hvor dem til kunden opstilles på et beskrivende niveau, som er forståeligt for andre end software udviklere. Derimod opstilles specifikationer på et mere detaljeret niveau til software udviklerne. [Sommerville, 2007, s. 75].

I udviklingsprocessen designes og programmeres det ønskede software udfra de stillede specifikationer. I design fasen laves en beskrivelse af software strukturen, hvilken data som skal håndteres og nogen gange snitfladerne mellem softwaren og andre komponenter. Yderligere udvikles flere modeller af systemet på forskellige abstraktionsniveauer, som hjælper med at klarlægge systemets del-komponenter. Dette vil sandsynligvis fremhæve hvis der er nogle uoverensstemmelser, som derfor kan rettes på et tidligt stadie. Efterfølgende programmeres softwaren, som indebærer løbende test af koden i de forskellige programdele i softwaren. Dette hjælper til at fejl rettes løbende, som dermed giver en lettere fejlsøgning, da det vides at det tidligere kode er testet og godkendt [Sommerville, 2007, s. 76-79].

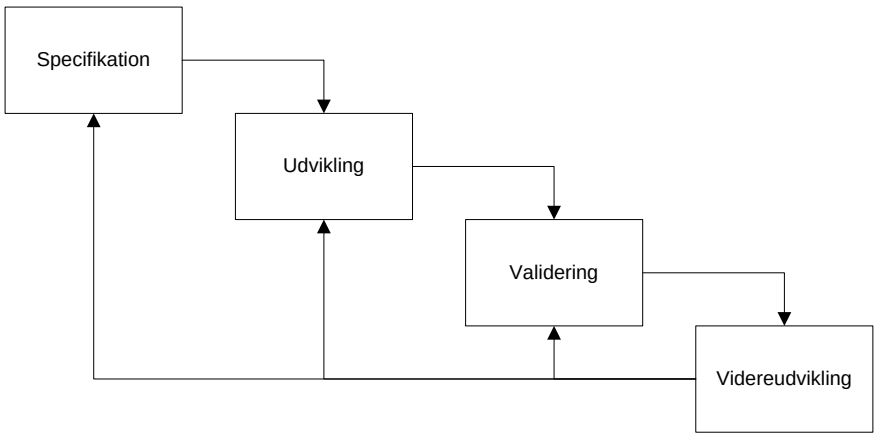
I valideringsprocessen udføres både en verificering og validering. I verificeringen kontrolleres om specifikationerne, som blev klarlagt i specifikationsprocessen, lever op til kundens forventninger. Valideringen består af at kunne vise, at softwaren udfører de funktioner som kunden forventer. Når større programmer skal valideres, er det en god idé ikke at afprøve hele programmet i en test, men i stedet gøre det i tre stadier, som er [Sommerville, 2007, s. 80-81,516]:

1. *Test af komponenter* - teste mindre program-dele/komponenter for at sikre disse opererer korrekt.

2. *Test af hele programmet* - komponenterne sammensættes og testes som én enhed, hvilket sikre at uventede fejl rettes, som kan opstå når komponenterne interagere og udføre de ønskede funktioner.

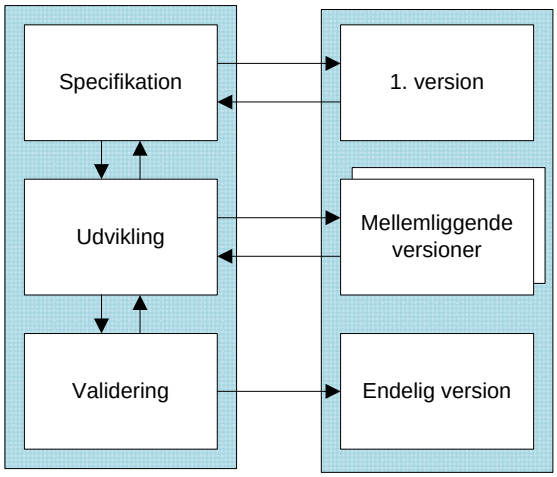
3. *Slut test* - programmet afprøves med data fra kunden selv, og ikke test data, hvilket kan medføre at programmet afsløre nye fejl. Slut testen fortsætter indtil udviklere og kunden er enige om det færdige program lever op til de opstillede krav.

De beskrevet processer kan gennemføres med forskellige procesmodeller, hvor de mest anvendte procesmodeller er vandfaldsmodellen og iterationsmodellen. I vandfaldsmodellen gennemføres hvert enkelt proces inden den næste proces påbegyndes, hvilket ses på figur 2.3. Fordelen ved at anvende vandfaldsmodellen er at dokumentation udføres i alle processerne, hvilket gør det nemmere at forstår de udførte opgaver. En ulempe kan være at opstillede krav i starten gør det sværere at lave ændringer, hvis kunden ønsker dette. En anden ulempe er, at kunden ikke har set en beta-version af softwaret inden det skal slut testes, hvormed ønskede ændringer i denne fase vil være svære at opfylde. I praksis anvendes vandfaldsmodellen med overlap, hvilket vil sige at der gives feedback til forrige proces, såsom hvis der i starten af udviklingen opstår problemer med specifikationerne, informeres dette tilbage og rettes [Alam, 2011] [Sommerville, 2007, s. 67].



Figur 2.3: Vandfaldsmodellen hvor hver proces gennemføres inden næste påbegyndes[Sommerville, 2007, s. 66].

I den iterative model udvikles en 1. version af det ønskede software ud fra de stillede specifikationer, hvor i større programmer udvikles der nogle få komponenter, som kan udføre nogle funktioner. Denne version fremvises til kunden, som giver feedback hvilket leder til ændringer i specifikationerne. Derved opstilles nye specifikationer, og mellemliggende versioner af softwaren udvikles og fremvises for kunden. Dette foregår indtil kunden er tilfreds og dermed er den endelig version udviklet. Denne model har den fordel at specifikationer, som er uklare for udvikleren bliver forståelige efterhånden som flere iterationer gennemføres, fordi kunden bliver mere klar over, hvilke opgaver softwaren skal løse. Ulempen ved at anvende iterationsmodellen er at dokumentation af softwaren ikke forefindes, da det ikke er økonomisk effektivt at dokumentere hver version. Dette medføre at projektlederen har svært ved at følge med i fremgangen af softwaren. En anden ulempe er at softwarens programkode ofte bliver ustruktureret pga. de løbende ændringer.



Figur 2.4: Iterationsmodellen [Sommerville, 2007, s. 67]

Ud fra beskrivelsen af disse procesmodeller, synes iterationsmodellen at være anvendt i det tidligere udvikling, som foregik på 9. semester. Her blev der udviklet en 1. version af simuleringen, hvilket havde det formål, at afprøve muligheden for at udvikle en simulering i C#, som kunne opfylde de stillede krav. Denne 1. version af simuleringen blev, som tidligere beskrevet, vist til OCTOMATION som godkendte konceptet af simuleringen og der blev stillet nogle yderligere specifikationer. Dermed vil der i det efterfølgende afsnit beskrives hvorledes specifikationer for udvikling af software opstilles og dermed opstille specifikationerne til simuleringen.

2.2.2 Specifikationer

Når software skal udvikles er det en god idé, at specificere hvilke krav der stilles. Disse krav opstilles på to forskellige niveauer, hvor kravspecifikationen til kunden er på et højt niveau og et mere detaljeret niveau, som anvendes af systemudviklerne. Disse kravspecifikationer kaldes hhv. for brugerkrav og systemkrav.

I brugerkravene beskrives den service, som det forventes at softwaren skal udføre. Dette beskrives i et naturligt sprog, som kunden kan forstå, hvormed det er vigtigt at der ikke indblandes noget programmeringsjargon. Dette kan være svært at undgå pga. at det ikke er muligt at præcisere kravene i et naturligt sprog.

Systemkravene er en udvidet version af brugerkravene, hvor systemkravene beskrives i et programmeringsteknisk sprog. Dette er vigtigt da en beskrivelse af kravene på et naturligt sprog kan misforståes. Dette skyldes at et naturligt sprog er overfleksibelt, hvor det samme ord kan forståes på forskellige måder. Til at opstille systemkravene anvendes der ofte nogle standardformularer, hvilket gør det nemmere for en systemudvikler at læse og forstå disse. Disse standardformularer kan består af følgende punkter, som ses i tabel 2.2 [Sommerville, 2007, s. 127-132]:

Navn på softwaren	
Funktion	Softwarens funktion
Beskrivelse	Beskrivelse af softwaren
Input og source	Beskrivelse af inputs og hvor disse kommer fra
Output og destination	Beskrivelse af outputs og deres videre virken
Aktion	Beskrivelse af hvorledes softwaren specifik udføre opgaven
Krav	Kravene for at udføre opgaven
For- og efterbetingelse	Hvis der stilles nogle betingelser, som skal være sandt inden funktionen udføres, og hvad der er sandt efter dette.
Sidevirkninger	Beskrivelse af eventuelle sidevirkninger ved opgaven der udføres af softwaren

Tabel 2.2: Her ses nogle punkter, som kan anvendes til at udføre en standard formular, der beskriver systemkravene.

En sådan standard formular udarbejdes, hvilket skal gerne skal klarlægge de funktion-

er som simuleringssystemt skal kunne udføre.

Simuleringssystem	
Funktion	Afprøve forskellige indstillinger i simuleringssystemet og efterfølgende udregne konsekvensen af indstillingerne mht. ydelsen for pakkeanlægget
Beskrivelse	I simuleringssystemt skal de være muligt at opsætte nogle ønskede indstillinger for pakkeanlægget, som efterfølgende skal simuleres. Nogle af de vigtigste indstillinger er: Antal robotceller; input rate for emner og kasser; Prioriteringen, som skal anvendes i de enkelte robotceller; Antal targets i kasser; Tilvalg af 2. sorteringsemner; Transportbåndenes hastighed; Efterfølgende skal det tydeliggøres hvorledes pakkeanlægget performede.
Input og source	Simuleringen bruger ingen datainput - alt genereres ud fra indstillingerne.
Output og destination	Pakkeanlæggets performance i form af f.eks. antal pakket og ubehandlet emner, antal færdig/ikke-færdig pakkede kasser, udnyttelsesgrad m.m., skal gemmes. Dette output skal anvendes i OCTOMATIONs statistik modul, som analyserer disse output og præsenterer pakkeanlæggets performance.
Aktion	Overordnet set skal simuleringssystemet generere emner og kasser udfra input rate m.m. Disse kører gennem pakkeanlægget på transportbåndene efter valgt hastighed på båndene. Hvis der er ledige emner og kasser i samme robotcelle, og robotten er ledig, skal emnerne pakkes i kasserne, efter gældende prioriteringsregler m.m.
Krav	-
For- og efterbetingelse	-
Sidevirkninger	-

Tabel 2.3: Simuleringssystemets systemkrav.

Dermed er der opstillet systemkrav for simuleringssystemet. Dette er ikke så specifikt, som ønsket, men da der arbejdes med en iterativ procesmodel kan systemkravene revideres og ændres løbende. Dette er en god idé hvis der er flere systemudviklere tilknyttet, og dermed vil de vide hvilken retning udviklingen tager.

2.2.3 Design af brugerflade

Det er vigtigt at implementere nogle designpricipper i brugerfladen, fordi en dårlig brugerflade vil medføre, at brugeren ikke kan benytte systemet fuldt ud og sandsynligvis vil lave fejl. Derved vil en bruger hurtigt opfatte at softwaren forhindrer i stedet

for at hjælpe til at løse en opgave. En generel designregel er at minimere antallet af informationer som bliver givet på brugerfladen, da menneskets korttidshukommelse kun kan rumme 7±2 informationer ad gangen. Derfor vil der sandsynligvis opstå brugerfejl hvis dette ikke overholdes, og især hvis brugeren er under stress [Sommerville, 2007, s. 363].

I tabel 2.4 ses nogle generelle designpricipper [Sommerville, 2007, s. 364]:

Princip	**Beskrivelse**
Bruger kendskab	Brugerfladen skal indeholde termer og koncepter, som er velkendte for de brugere, som skal anvende systemet mest
Kontinuitet	Brugerfladen skal være konsistent forstået på den måde, at sammenlignelige operationer skal udføres på samme måde
Minimal overraskelser	Brugeren skal aldrig blive overrasket over hvordan systemet opfører sig
Genanvendelighed	Brugerfladen skal indeholde muligheder for at fortryde den valgte operation
Bruger vejledning	Brugerfladen skal give feedback hvis en fejl opstår og hjælp til løsning af dette
Bruger forskellighed	Brugerfladen skal have forskellig muligheder ift. hvilken bruger der anvender systemet

Tabel 2.4: Her ses generelle designprincipper, som det anbefales at bruge til design af en brugerflade.

Udover disse designprincipper, er det også vigtigt at fastsætte hvordan brugeren skal interagere med softwaren. Dette kan gøres vha. *Direkte manipulation*, *Menu selektion*, *Formular*, *Kommando sprog* og *Naturligt sprog*. Disse måder at interagere har både fordele og ulemper, hvor den interaktionsmetode, som vil passe bedst til en simulering er Menu selektion. Dette skyldes at simuleringen vil have en del valg i form af f.eks. input rate, antal robotter osv., som er rimelig fastlagt og derfor vil Menu selektion være en god måde at præsentere disse valg på. De fordele som Menu selektion giver er minimal indtastning, som derved minimere brugerfejl. Ulempen ved at anvende Menu selektion er, at det kan være for langsomt til nogle erfarne bruger af softwaren, samt hvis der er for mange menuer, kan det hurtigt blive for komplekst og uoverskueligt at anvende [Sommerville, 2007, s. 367].

Yderligere er der nogle designprincipper mht. antal farver og hvorledes informationer skal fremvises på brugerfladen. Det er vigtigt ikke at anvende for mange farver og heller ikke for lyse farver, da det gør brugerfladen forvirrende at se på. Det er en god idé at maksimalt bruge fire til fem farver på et vindue, og maksimalt syv farver i alt på hele brugerfladen. Yderligere kan det være en fordel at anvende et farveskift, hvis der sker et signifikant event. Der skal også være kontinuitet mht. farverne, således at f.eks. rød betyder fejl i alle vinduer [Sommerville, 2007, s. 374]. Når informationer på en brugerflade skal præsenteres, er det en god idé at holde informationen seperat fra den måde informationen bliver fremvist på. Denne måde gør det er muligt, at udskifte det modul som præsenterer informationerne, hvilket i dette tilfælde er OCTOMATIONs statistik modul [Sommerville, 2007, s. 370].

2.2.4 Dokumentation af software

Det ønskes at få klarlagt, hvordan arkitekturen af software kan dokumenteres på en overskueligt måde. En standard metode som oftest anvendes er UML, som står for Unified Modeling Language. UML består af flere typer diagrammer, som har til formål at danne et overblik over softwarens objekter.

Sekvens diagrammer danner et overblik over, hvorledes objekterne interagere i en sekventiel rækkefølge. Denne metode ønskes anvendt til at dokumentere simuleringen, og derfor gives en introduktion til brugen af sekvens diagrammer.

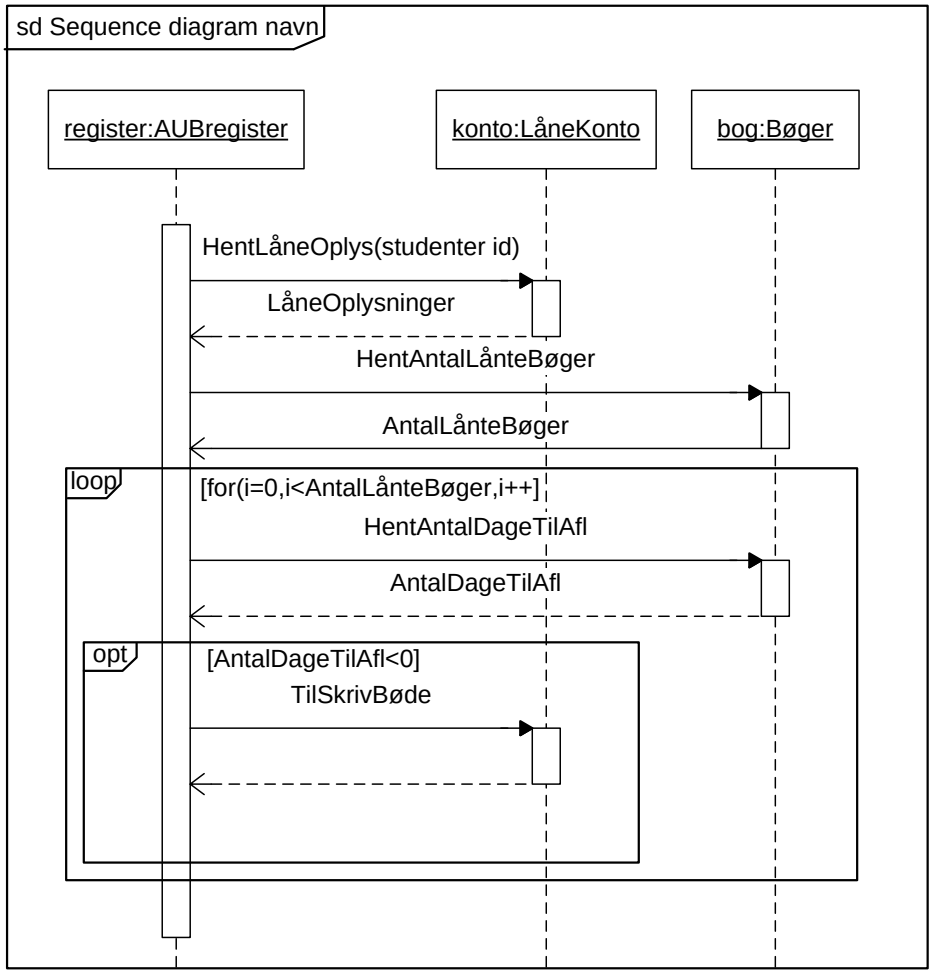
Et sekvens diagram består af en ramme omkring sekvenserne og i venstre hjørne angives navnet på diagrammet. Diagrammet giver informationer både horisontalt og vertikalt: Horisontalt vises objekter; Vertikalt, fra top til bund, er rækkefølgen hvormed sekvenserne bliver eksekveret.

Objekterne er angivet med klassetype og navnet på objektet - dette skrives *navn : klassetype*. På objektet er der en livline, som vertikalt går ned gennem diagrammet. Et objekt er kun aktivt, når der på livlinjen er placeret et rektangel, hvor fra der sendes og modtages beskeder. Når et objekt sender en besked til et andet objekt, markeres dette med en line og et solidt pilhoved, og en retur besked markeres med en stiplet line og et spidst pilhoved.

I sekvens diagrammer defineres if-sætninger med *opt*. Dette gøres ved at lave en ramme og skrive opt i venstre hjørne. Til højre herfor skrives den betingelse, som skal være opfyldt hvis de sekvenser indenfor opt-rammen skal eksekveres. Ligeledes gøres for loops, hvor sekvenserne indenfor loop-rammen eksekveres indtil betingelsen for loopet

er udført.

På figur 2.5 på modstående side ses et eksempel på et sekvens diagram, som skal vise en del af et bibliotek system. Der tages udgangspunkt i en programdel, som skal kontrollere om lånere aflevere bøger for sent, og derfor skal tildeles en bøde. Programdelen består af tre objekter *AUBregister*, *LåneKonto* og *Bøger* á klassetyperne *register*, *konto* og *bog*. Programdelen starter med at indhente oplysninger om biblioteksláneren med metoden *HentLåneOplys* som kræver *studenter id*, indhenter oplysningen fra LåneKonto og retuner låneoplysningerne. Derefter hentes det antal bøger, som er lånt på denne konto, og hvis dette antal er større end nul, eksekveres for-loopet, som indeholder sekvenserne vist i rammen *loop*. Dette for-loop eksekveres det antal gange, som der er lånt bøger. I for-loop'et er en if-sætning, *opt*, som kun eksekveres, hvis bogens afleveringsdato er overskredet, hvilket medfører at der sendes en bøde til den pågældende lånekonto [Bell, 2004].



Figur 2.5: Her ses et eksempel på et sekvens diagram, som indeholder tre objekter, en if-sætning (opt) og et for-loop.

En anden metode at dokumentere hvorledes et program eller en metode fungerer, er Pseudo kode. Pseudo kode er en måde at skrive program kode på, som ikke specifikt anvender termer fra programmeringssprog, såsom C#, C++ osv. Dette gør Pseudo kode til en metode der generalisere programkode, som programmører kan oversætte til forskellige programmeringssprog [Nishimura, 2011]. I algoritme 1 på næste side skrives eksemplet fra tidligere som pseudo kode.

Ud fra algoritme 1 på den følgende side kan det ses at dette programkode ikke indeholder nogle termer fra andre programmeringssprog, men det er muligt at oversætte det til disse, da det er synligtgjort at der skal indgå et for-loop og en if-sætning.

Algorithm 1 Bøde tjek

```
1: Hent låner oplysninger med studenter id
2: Hent antal lånte bøger
3: for (i=0, i<AntalLånteBøger, i++) do
4:     Hent antal dage til aflevering
5:     if Antal dage til aflevering < 0 then
6:         Tilskriv bøde til konto
7:     end if
8: end for
```

2.3 Skedulering

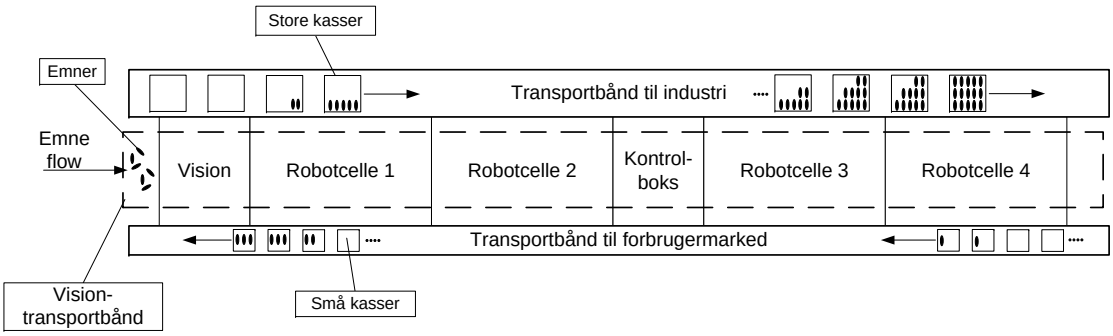
I dette afsnit gives et overblik over flowet af emner og kasser i anlægget og hvilke udfordringer dette giver. Herefter undersøges det, om det er muligt at udarbejde en styring som bygger på realtidsskedulering.

Som tidligere beskrevet i afsnit 1 på side 1 består anlægget hos Rahbek af et visionsystem, tre transportbånd og fire robotceller, som har en Adept Quattro s650H robot placeret centralt heri. Udfordringen består af at få disse komponenter til at fungere i et sammenspil, som giver den bedste ydelse mht. antal pakket kasser. Det kan ses på figur 2.6 på modstående side at emnerne kommer ind på visiontransportbåndet, hvor de bliver detekteret under visionsystemet, således at emnernes placering bliver kendt. Efterfølgende skal emnerne pakkes i enten kasser til forbrugermarkedet eller til industrien. Den største udfordring er at pakke emnerne til forbrugermarkedet, da kasserne her kører modsat af visiontransportbåndet, hvilket kaldes crossflow. Dette crossflow er lavet for at sikre at emnerne til forbrugerne er ude af fryseren i kortest mulig tid. Der pakkes ikke emner til forbrugermarkedet og industrien samtidig. Begge bånd bliver dog stadig brugt, når der pakkes til forbrugermarkedet, da anden sorteringsemner bliver pakket i de store kasser[OCTOMATION, 2011].

På figur 2.7 på side 18 ses netop crossflowet mellem de to transportbånd. Emnerne på visiontransportbåndet er på vej ind under visionsystemet, og der kommer pakkede kasser med emner ud i venstre side.

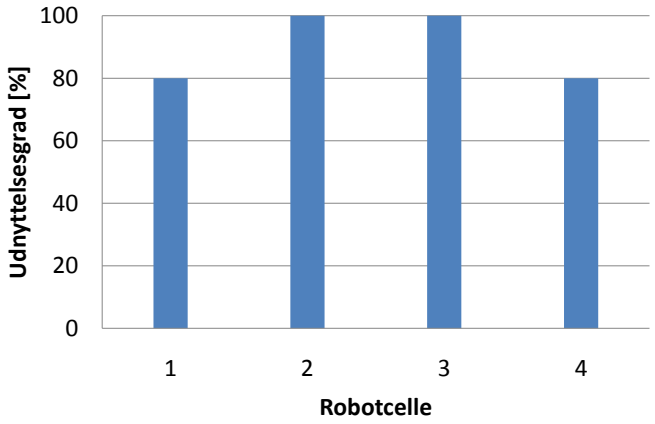
I tabel 1.2 på side 4 blev de kriterier, som ønskes minimeres og maksimeres opstillet. Disse er, som tidligere beskrevet, afhængige af hinanden, og derfor klarlægges nu problemstillingen mht. at minimere eller maksimere disse.

Forskellen på robotternes udnyttelsesgrader er høj, hvilket figur 2.8 på næste side viser. Hvor de centrale robotter næsten er på 100%, så er de yderlige robotter nede omkring



Figur 2.6: Her ses anlægger fra oven, bestående af visionsystemet, de tre transportbånd og de fire robotceller.

80%. For Robot 1 skyldes det, at det er dennes opgave at fylde de sidste kasser helt op - med den nuværende styring vil den engang imellem stå og vente på en kasse, som faktisk har plads til emner. For Robot 4 skyldes den lavere udnyttelsesgrad, at det er dennes opgave at sikre, at de sidste emner på båndet bliver pakket, inden de kører ud af anlægget. For at sikre, at den altid vil kunne opfylde dette, kommer der færre emner til den - den har altså en slags sikkerhedsbuffer, så man er sikker på, at den kan følge med [OCTOMATION, 2011]. Denne strategi opfylder derfor kriterierne om at minimere antallet af ubehandlet emner og ikke-færdig pakkede kasser, men strider imod at maksimere den homogene fordeling af arbejdsbelastningen mellem robotterne.



Figur 2.8: Udnyttelsesgraden for anlægget - de centrale robotter arbejder så godt som konstant, mens de yderliggende robotter har mindre at lave. Dette er nødvendigt for bl.a. at sikre, at samtlige emner og kasser bliver pakket inden de forlader anlægget.

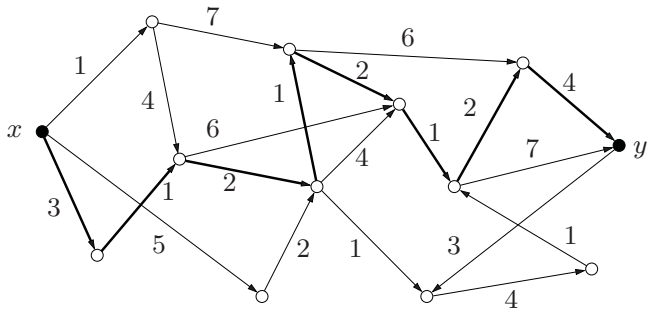
Yderligere ønskes det at maksimere antallet af pakkede emner og kasser, hvilket kun kan lade sig gøre ved at øge den samlede udnyttelsesgrad på anlægget. Dette vil sandsyn-

Figur 2.7: Pakkede kasser ses på transportbåndet til venstre, som har retning mod bunden af billedet og emnerne ses til højre på visiontransportbåndet med modsat retning [OCTOMATION, 2011].

ligvis medvirke til, at der opstår en konflikt i form af, at antallet af ubehandlet emner og ikke-færdig pakkede kasser stiger. Til at løse den problemstilling kan skedulering måske hjælpe, så der kan planlægges hvilke robotter, der skal behandle hvilke emner og hvilken kasse disse emner skal placeres i. Én skeduleringsmetode er Shortest Path, som vil blive gennemgået i det følgende afsnit.

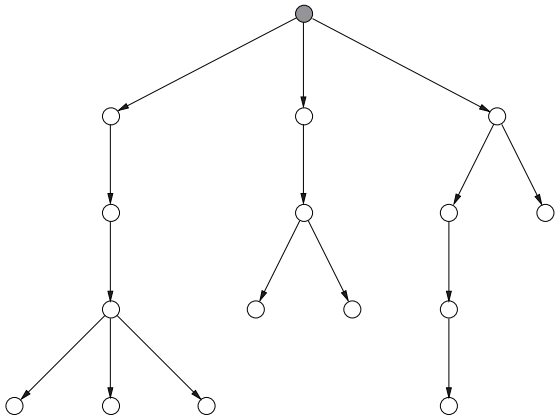
2.3.1 Shortest Path

Shortest Path bliver bl.a. brugt i Graf Teori, hvor den anvendes til at finde den korteste rute på en graf. En graf skal forståes som nogle knudepunkter, hvor nogle linjer forbinder disse. Shortest Path anvendes til at finde den korteste rute mellem to knudepunkter. Disse knudepunkter x og y kan f.eks. være kommunikationscentre og linjerne mellem disse er ledningsnettet, hvormed det vil være interessant at finde den korteste mulige kommunikationsvej mellem x og y, som kan ses på figur 2.9 på modstående side[Bondy and Murty, 2008, s. 150].



Figur 2.9: Her ses en graf bestående af knudepunkter og linjer. Linjerne har en afstand og Shortest Path beregner alle mulighederne for at komme fra x til y, og giver den korteste rute mellem x til y [Bondy and Murty, 2008, s. 150].

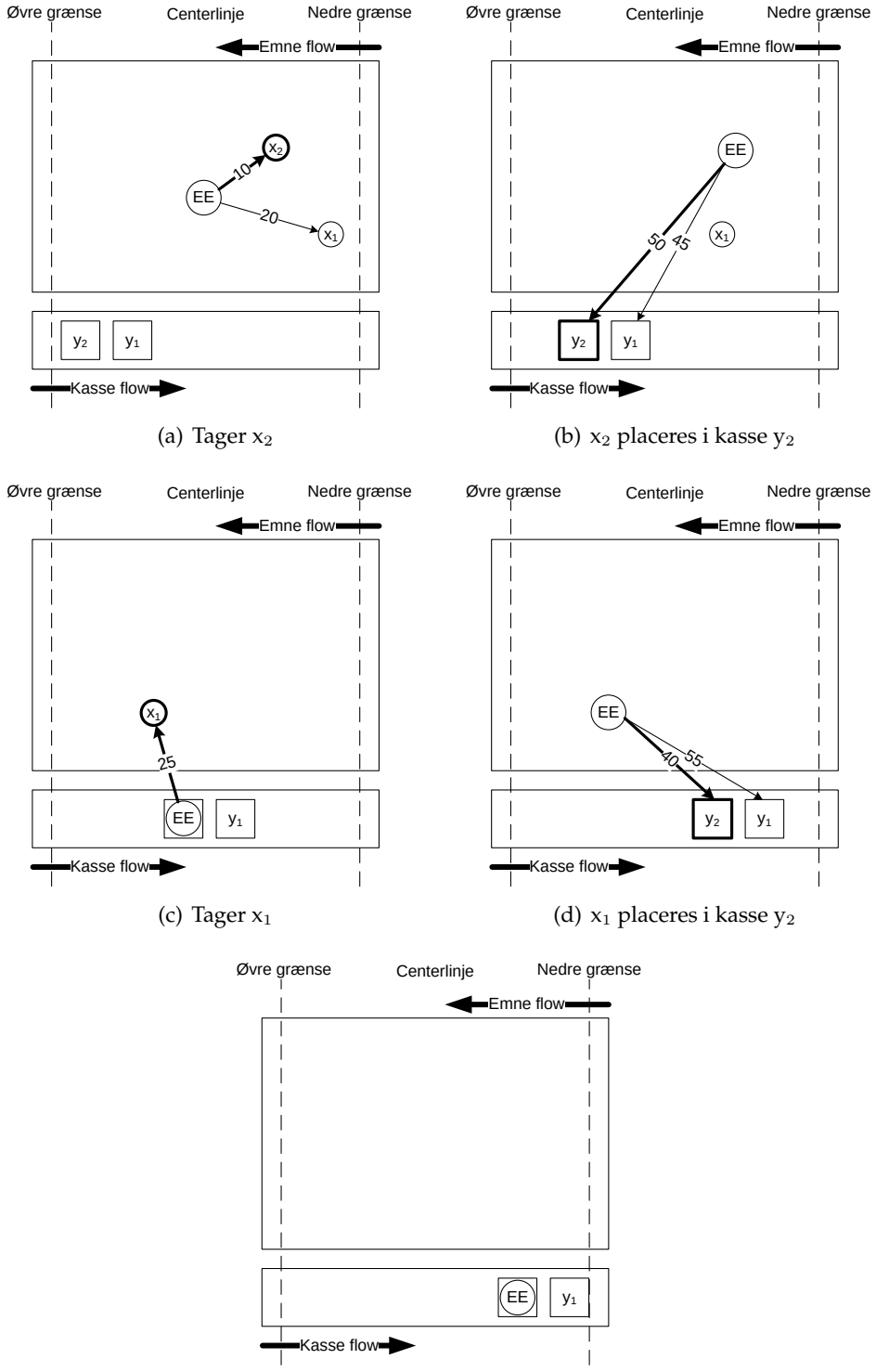
Ud fra en sådan graf er det muligt at opbygge et træ, som danner et bedre overblik over grafen. Dette gøres ved at vælge et start knudepunkt, som f.eks. x, og derefter danne en trækrone, hvor hver gren repræsentere en løsning for at komme fra x til y. På figur 2.10 ses et eksempel på et træ [Bondy and Murty, 2008, s. 100].



Figur 2.10: Her ses et eksempel på et træ, som f.eks. kunne repræsentere alle løsninger for at komme fra x til y.

Denne metode kan være interessant at implemetere i pakkeanlægget, hvor knudepunkterne er emner og kasser og linjerne er procestiderne mellem disse. Det er klart, at der skal tages højde for at pakkeanlægget er et dynamisk system, og dermed vil det kræve, at der vælges et bestemt antal af emner og kasser, som inden for en tid befinder sig i pakkeanlægget og indenfor robotternes arbejdsområder, hvilket kan gøres ud fra visionsystemet. Derved kan der ud fra disse udregnes alle de mulige ruter, som robotterne kan vælge for at pakke emnerne i kasserne. Udfra disse udregninger vælges da den hurtigste rute, Shortest Path, og operationerne i denne udføres.

Det er ikke muligt at optegne en graf af sådan et system pga. dynamikken, da knudepunkterne flytter sig i forhold til transportbåndenes hastighed, og dermed ændres linjerne (procestider) mellem disse alt efter hvilket objekt der behandles. Yderligere skal robotterne vælge mellem to typer knudepunkter, emner og kasser, alt efter hvilket stadie robotterne befinder sig i - der skal sørges for at en rute fra f.eks. kasse til kasse ikke bliver en løsning. På figur 2.11 på modstående side er de forskellige stadier i et pakke scenarie illustreret. Robotten skal igennem fem knudepunkter, bestående af to emner, to kasser og robottens End-Effector. End-Effectoren skal følge sekvensen med at først tage et emne og komme dette i en kasse.



(a) Tager x_2

(b) x_2 placeres i kasse y_2

(c) Tager x_1

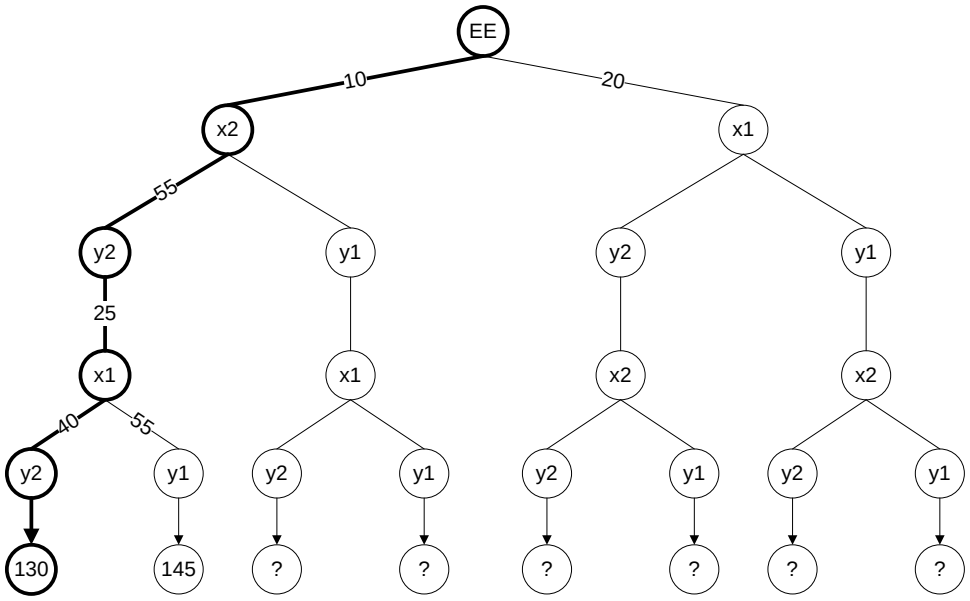
(d) x_1 placeres i kasse y_2

(e) Operation udført

Figur 2.11: Her ses en illustration af Shortest Path, hvor det ses at der skal tages højde for dynamikken ved udregning af hver gren.

Dermed kan der opbygges et træ af alle løsninger, som der ses på figur 2.12. På et sådan træ kan der anvendes nogle søge teknikker fra Graf teori, som f.eks. søger alle løsningerne på træet igennem og giver på den korteste rute.

I dette tilfælde er det dog ikke muligt at anvende disse søge teknikker, da dynamikken gør at procestiderne ændres under en sådan søgning. Derfor skal alle løsninger udregnes alt efter hvilket emne eller kasse der vælges, og herefter kan Shortest Path vælges. På figur 2.12 ses løsningen af tilfældet på figur 2.11 på foregående side, hvor ruten udregnes til 130.



Figur 2.12: Her ses et træ, hvor den yderste gren til venstre repræsentere løsningen af scenariet der tidligere blev vist.

Denne skeduleringsmetode vil være meget interessant at få implementeret, men da der konstant befinder sig hundreder af emner i anlægget, vil det kræve en meget stor udregning at finde den optimale løsning. Dette ville desuden kræve en ret så stor CPU kraft, hvilket ikke vil være tilgængelig med den nuværende controllerboks. Dette skal især ses i lyset af, at det ville kræve en reskedulering meget ofte - hver robot er kun ca. et halvt sekund om at pakke et emne og 5-10 nye emner vil komme ind i anlægget hvert sekund. Det vil derfor kræve en meget hurtig søge algoritme for at finde den bedste løsning - en søge algoritme som hverken er til rådighed eller mulig at udarbejde for denne projektgruppe. Derfor vurderes denne løsning som ikke mulig.

Derfor vil det blive forsøgt at finde andre metoder til at udarbejde en pakkestrategi.

2.4 Pakkestrategi

Til at løse en problemstilling, som er så kompleks at den optimale løsning ikke kan findes inden for et bestemt tidsrum, anvendes heuristik [Diaconis, 2003, s. 36]. Heuristik anvendes til at finde hurtig løsning, som stadig er fornuftig, hvilket er anvendeligt til problemstillingen omkring pakning af alle emnerne og kasser i pakkeanlægget. Heuristiske prioriteringsregler anvendes til at prioritere den rækkefølge, som job behandles på ved hver enkelte arbejdsstation. Dette giver en hurtig løsning, da det på hver enkelt arbejdsstation altid er kendt hvilke job der skal behandles og i hvilken rækkefølge [Stevenson, 2007, s. 732]. Udover heuristiske prioriteringsregler vil en retningsbestemt prioritering også give fordele til at kontrollere, hvilke robotter der får de mest tidskrævende opgaver. Desuden kan arbejdet fordeles mellem robotterne ved bl.a. at opsætte begræsninger for, hvor mange emner de enkelte robotter må pakke i kasserne.

Dette giver i alt en pakkestrategi, som består af følgende:

Regel	**Beskrivelse**
Heuristisk prioritering	Emnerne i robotternes arbejdsområde opsættes i en liste og sorteres efter regler som ankomsttid og procestid -FIFO, LIFO, SPT, LPT m.fl.
Vertikal prioritering	Emner prioriteres efter, hvor langt væk fra kassebåndet de befinder sig
Antal frie targets	Hvor mange frie pladser skal der minimum være tilbage i en kasse, når den forlader arbejdsområdet

Tabel 2.5: Pakkestrategierne i simuleringen vil blive opbygget af disse regler.

Disse regler vil ikke give den optimale løsning, men kan give løsninger som er tilstrækkelig gode. Brugerne af simuleringssystemet vil udfra disse regler kunne sammensætte forskellige pakkestrategier, som passer til forskellige anlæg.

De overstående regler vil i de følgende afsnit blive uddybet.

2.4.1 Prioriteringsregler

Prioriteringsregler skal forståes som den rækkefølge objekter prioriteres efter, når de skal behandles. Prioriteringsreglerne er:

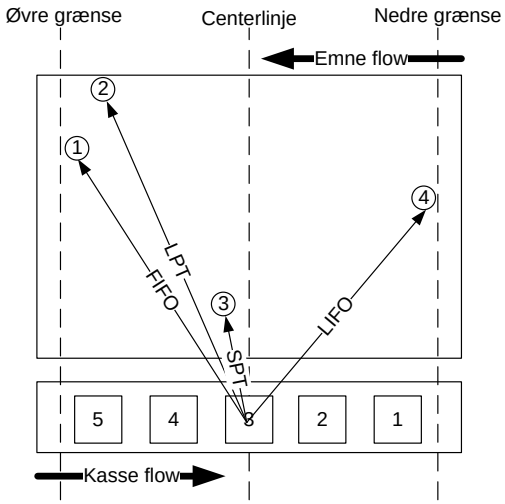
- FIFO - First In First Out
- LIFO - Last In First Out

- SPT - Shortest Processing Time
- LPT - Longest Processing Time
- Modificeret SPT
- Forbedret SPT

FIFO, LIFO, SPT og LPT

Ved FIFO bliver det første tilkomne objekt behandlet først, som kan sammenlignes med en kø i et supermarked og med LIFO bliver det sidste tilkomne objekt behandlet først, hvilket kan ses på figur 2.13 [Stevenson, 2007, s.732]. Dvs. at med FIFO vil robotten pakke det emne, som er kørt længst hen ad visionbåndet - med LIFO vil det emne, som har kørt kortest på båndet blive valgt.

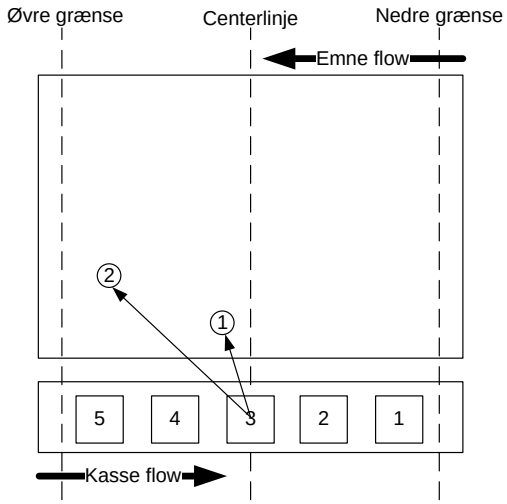
SPT og LPT er anerledes i forhold til FIFO og LIFO på den måde, at de udføres i forhold til den tid, det tager at behandle objekterne. Dette vil sige at SPT vil prioritere det emne, som tager kortest tid og LPT vil prioritere det emne, som tager længst tid at behandle, hvilket kan ses på figur 2.13 [Stevenson, 2007, s.732]. Behandlingstiden skal i dette tilfælde forståes som den tid, som robotten bruger på at udføre en pick-and-place operation.



Figur 2.13: Her ses FIFO, LIFO, SPT og LPT i forhold til at robottens End-effector befinder sig over en kasse ved centerlinjen.

Modificeret SPT

Modificeret SPT tager højde for, at der i nogle tilfælde vil være objekter, som bør vægtes højere end det objekt med den korteste procestid. Dette er eksempelvis når et objekt nærmerer sig den øvre grænse for arbejdsområdet, som ses på figur 2.14.



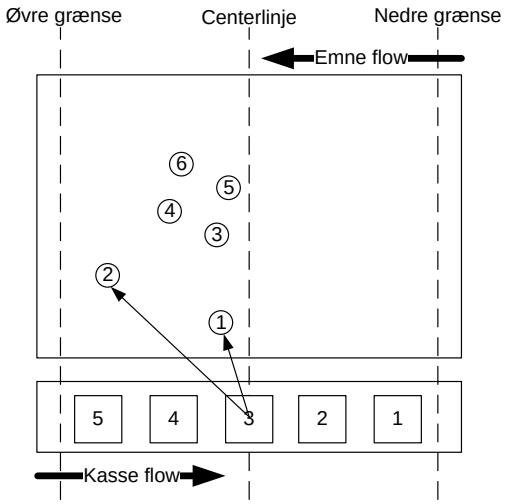
Figur 2.14: Objekt 2 nærmere sig øvre grænse for arbejdsområdet, hvilket medfører at Modificeret SPT vil prioriterer dette objekt højere end Objekt 1, som ellers har kortest procestid.

Modificeret SPT prioriterer efter det antal tilgængelige emner, som er tilbage efter at have behandlet et emne. Hvis Objekt 1 og Objekt 2 på figur 2.14 defineres som hhv. O_1 og O_2, kan betingelserne for Modificert SPT defineres således [Mattone et al., 1999, s.88-89]:

- *O_1 har kortest procestid, O_2 har anden kortest procestid*
- *n1 er det antal objekter, som er tilgængelige efter O_1 er behandlet, og S1 er det antal objekter, der forlader arbejdsområdet, hvis O_1 behandles*
- *Hvis S1 indeholder objekter, beregn da ny sitation, hvor O_2 bliver behandlet istedet for O_1. n2 er da det antal objekter, som er tilgængelige efter O_2 er behandlet*
- *Hvis n1 $\geq$ n2, sættes $O_{næste} = O_1$ ellers $O_{næste} = O_2$*

Denne definition medfører at det maksimale antal objekter efterfølgende er tilgængelig, når $O_{næste}$ vælges. Denne prioriteringsregel giver dermed en lokal forbedring, som dog ikke nødvendigvis medfører en global forbedring. Dette skal forståes på den måde, at

selvom der er flere tilgængelige objekter betyder det ikke nødvendigvis, at flere objekter bliver behandlet. Et eksempel på dette er situationen, som er vist på figur 2.15. Hvis O_1 behandles, vil O_2 kører ud af området, så der efterfølgende vil være 4 objekter tilgængelig. Behandles O_2 derimod, vil dette efterlade 5 objekter at behandle. Problemet er dog så at pga. O_2's lange procestid, vil de resterende emner være kørt så langt ned af båndet, at der højst sandsynligt kun kan behandles ét emne, inden de alle er kørt ud af området. Var O_1 derimod blevet behandlet med den korte procestid, ville O_3 og O_5 formentlig kunne behandles efterfølgende - dvs. ét mere emne, end hvis M-SPT reglerne blev fulgt.



Figur 2.15: Behandles O_1 vil der efterfølgende være 4 emner at behandle efterfølgende. Vælges det at behandle O_2 først, vil der være 5 emner tilbage.

En løsning på dette kunne være at O_2 kun vælges, hvis O_1 efterfølgende kan behandles med en kortere procestid efterfølgende, hvilket tages højde for i *Forbedret SPT* [Mattone et al., 1999, s.88-89].

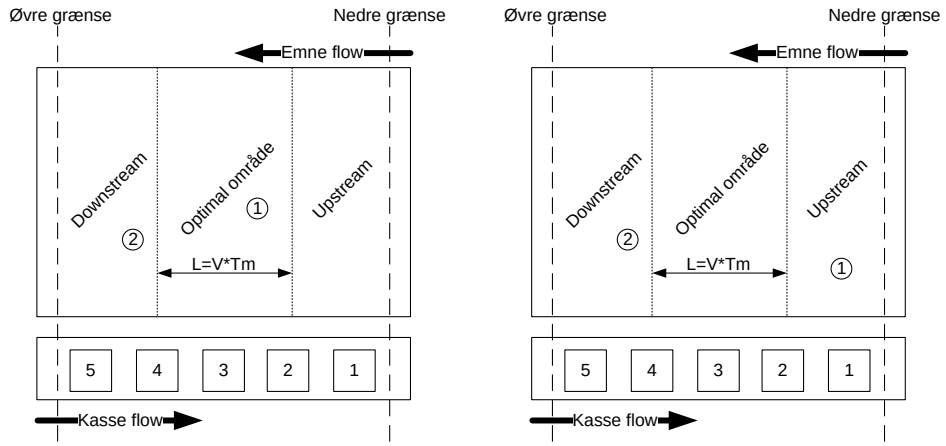
Forbedret SPT

Forbedret SPT giver mulighed for at behandle objekter, som nærmer sig den øvre grænse for arbejdsområdet. Dette kan lade sig gøre ved at inddele arbejdsområdet i områderne *Upstream*, *Optimalt område* og *Downstream*, som det ses på figur 2.16 på næste side. Det optimale område defineres ved at bredden er $L = V * Tm$, hvor V er transportbåndets hastighed og Tm er den maksimale pick-and-place procestid. Tm bestemmes som bevægelsen fra det ene hjørne i robottens arbejdsområde til det diago-

nale modsatte hjørne og tilbage. Forbedret SPT kan defineres således [Mattone et al., 1999, s.88-89]:

- *Lad O_1 være det næste SPT objekt og O_2 være i Downstream området. Hvis O_1 ikke befinder sig i Upstream området, så tag O_1, ellers tag O_2 i downstream området.*

Forbedret SPT indskrænker antallet af objekter, som bliver behandlet udover O_1. Dette betyder at objekter som ikke bliver behandlet inden Downstream området, har en risiko for ikke at blive behandlet, hvilket er en del af prioriteringsreglen. Hvis O_2 skal behandles i downstream området, er reglen at O_1 efterfølgende skal kunne behandles inden dette forlader det optimale område. Forbedret SPT skal ses som en fleksibel almindelig SPT, som har mulighed for at tage emner tæt på den øvre grænse for arbejdesområdet, når det ikke forlænger procestiden for næste emne.



(a) Her prioriteres O_1 fremfor O_2 (b) Her prioriteres O_2 fremfor O_1

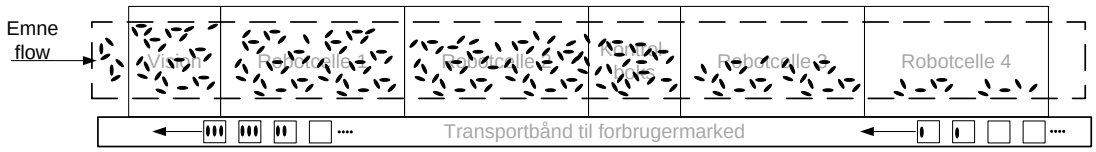
Figur 2.16: Forbedret SPT inddeler arbejdsområdet i tre områder: Upstream, Optimalt område og Downstream.

2.4.2 Vertikal prioritering

Udover at have prioritering ifht. procestid og hvornår emnet er ankommet i arbejdsområdet, så ville det også være en fordel, at kunne prioritere emnerne ifht. hvilken side af båndet de befinder sig på. Altså om de befinder sig tættest på det store eller lille kassebånd. Dette vil blive kaldt vertikal prioritering.

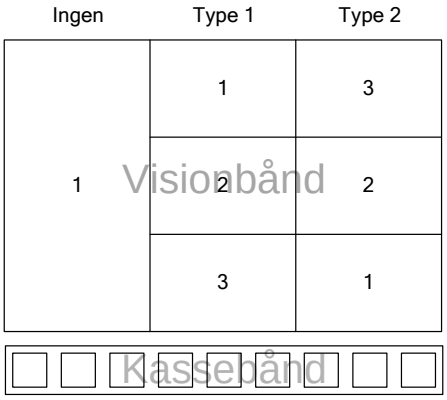
Vertikal prioritering kan være en fordel, når man gerne vil sikre sig, at den sidste robot på linjen kan nå at pakke alle emner, inden de ryger ud af anlægget. Hvis man f.eks.

får de centrale robotter til at tage de emner, som ligger længst væk fra kasserne, vil den sidste robot være sikret alle de emner, som ligger tættest på kasserne, og som dermed har mindste procestider, hvilket kan ses på figur 2.17.



Figur 2.17: Her ses et eksempel hvordan vertikal prioritering kan være en fordel når robot 1-3 tager emner længst væk og sikrer, at robot 4 kan nå at behandle alle emnerne tæt på kasserne inden de forlader pakkeanlægget.

Vertikal prioritering vil i praksis kunne laves således, at arbejdsområdet kunne deles op i tre vertikale områder, som vist på figur 2.18. Den type vertikal prioritering man vælger, vil da bestemme den rækkefølge, som de tre områder vil blive undersøgt i, når robotten søger efter ledige emner.



Figur 2.18: Med det lille kassebånd forneden og visionbåndet foroven, er der her illusteret, hvordan emnerne på visionbåndet kan prioriteres i den vertikale retning. Der er to typer: Type 1 - Her prioriteres emnerne længst væk fra kassebåndet. Type 2 - Her prioriteres emnerne tættest på kassebåndet.

2.4.3 Minimum antal targets

En metode, som kan være med til at fordele arbejdsbelastningen mellem robotterne, er at opsætte nogle regler mht. det antal frie pladser, targets, der minimum skal være tilbage i en kasse, når den forlader robotcellen. Man kan f.eks. indføre, at Robotcelle 2 minimum skal efterlade to frie targets og dermed sikre sig, at Robotcelle 1 altid har mindst to targets per kasse, som skal fyldes op med emner.

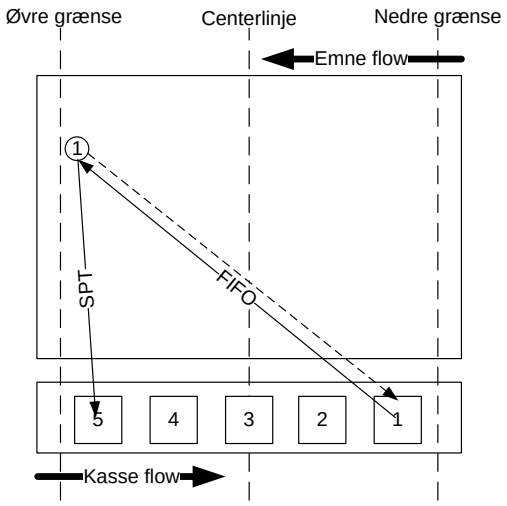
Sådan én regel vil være god til at fordele arbejdet i de tilfælde, hvor den samlede belastning på anlægget er lav. I disse tilfælde ville de centrale robotter næsten selv kunne fylde kasserne og dermed intet arbejde efterlade til den første robot i rækken. Det ville give en meget lav udnyttelsesgrad for denne første robot. Bruger man reglen om minimum targets bliver arbejdet altså fordelt bedre.

Man kan altså på den måde sikre, at den første robot er beskæftiget med konstant at pakke 1. prioriteringsemner. Den tid, som de centrale robotter får til overs med denne begrænsning, kan de så bruge på at pakke 2. sorteringsemner. En sidste konsekvens fra dette er, at det efterlader flere 1. prioriteringsemner til den sidste robot at pakke.

2.4.4 Diskussion ang. implementering af heuristiske prioriteringsregler

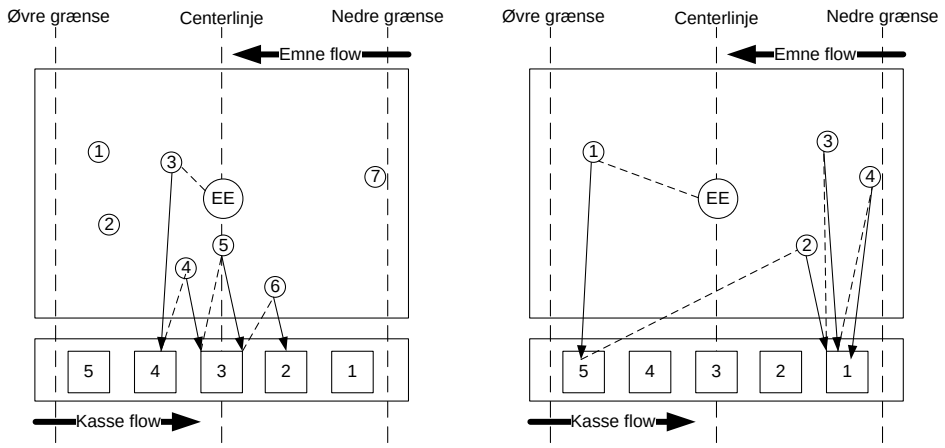
I det overstående blev de forskellige heuristiske prioriteringsregler beskrevet, hvor der i dette afsnit diskuteres muligheden for at anvende dem i pakkeanlægget. Alle prioriteringsreglerne anvender et fast punkt til at afsætte et objekt, hvilket ikke er tilfældet i pakkeanlægget. I pakkeanlægget kommer der to typer objekter, som er emner og kasser. Emnerne behandles ved at robotten griber og flytter dem til frie targets i en kasse. I pakkeanlægget er flow retningen, som tidligere beskrevet, på visionbåndet og kassebåndet modsatrettede, hvilket betyder at prioriteringsreglerne skal ses i forhold til om robottens End-effector skal bevæge sig til et emne eller til en kasse.

Hvis FIFO og LIFO anvendes vil det betyde at robotten hele tiden kommer til at udføre nogle lange bevægelser. Dette skyldes ved FIFO at det første indkomne emne altid skal pakkes i den første indkomne kasse, som befinder sig længst væk, hvilket ses på figur 2.19 på den følgende side. Det samme er tilfældet for LIFO, som vil udføre de lange bevægelser i modsat række følge, altså tage det sidste indkomne emne og komme i den sidste indkomne kasse. I stedet for kun at anvende FIFO og LIFO i begge retninger, kunne det være en fordel at f.eks. anvende SPT, når robotten skal bevæge sig mod en kasse og derved minimere afstanden. Denne fleksibilitet er vigtigt at få indført i simuleringen, således disse valg kan tilvælges - man skal altså kunne vælge prioriteringsregel både for emner og kasser.



Figur 2.19: Her ses End-effectoren ved kasse 1, og i følge FIFO skal den bevæge sig til det første emne og efterfølgede til første kasse indenfor arbejdsområdet, som sandsynligvis er kasse 1 eller 2, hvilket vil give store afstande som robotten skal bevæge sig. I stedet kunne SPT anbefales når robotten skal fra emne til kasse, og minimere afstanden.

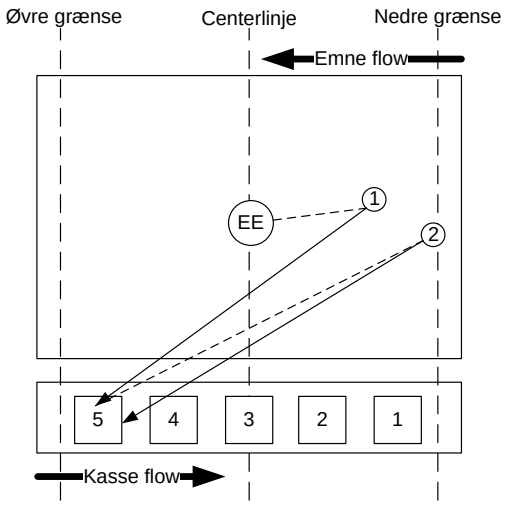
Prioriteringsreglen SPT vil altid sørge for at det emne, der er tættest på robottens End-effector, behandles først og kommer det i den nærmeste kasse. Hvis belastningen bliver høj og over det maksimale, som robotten kan nå at behandle, vil det medføre at robotten vil blive i det samme område, og lade nogle emner forbigå. Dvs. at hvis den starter centralt, vil den forblive centralt, så længe der er nok emner. Ved lav belastning medfører SPT, at robotten vil bevæge sig mod den nedre grænse for arbejdsområdet og vil derefter blive ved med at opererer i dette område, hvilket kan ses på figur 2.20.



(a) SPT ved høj belastning (b) SPT ved lav belastning

Figur 2.20: SPT ved høj belastning (t.v.) vil medfører, at End-effectoren vil holde sig centralt, da disse emner vil være hurtigst at behandle og der kommer konstant flere. Ved lav belastning (t.h.) vil SPT medfører, at End-effectoren søger mod den nedre grænse, da den vil kunne tage emnerne lige så snart de kommer ind i arbejdsområdet

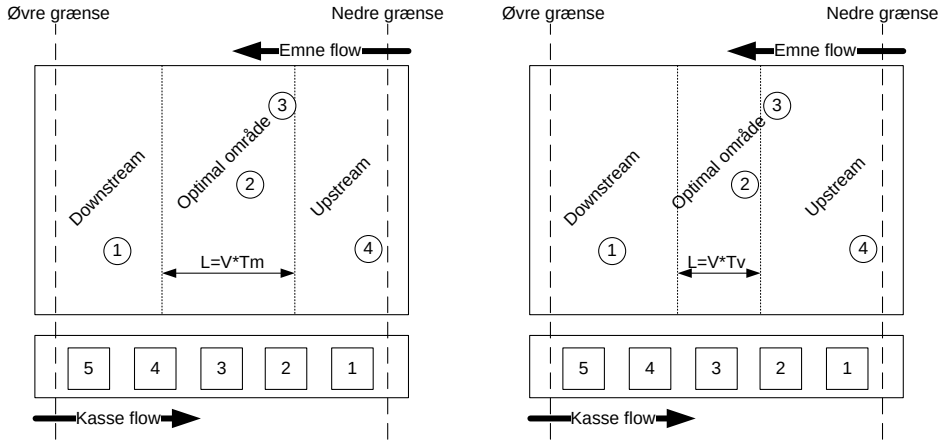
LPT vil ligesom SPT efter noget tid danne et bestemt mønstre hvormed emnerne behandles, alt efter hvor stor belastning der er på anlægget. LPT vil altid medføre at robotten laver nogle lange diagonale bevægelser, se figur 2.21 på den følgende side. Ved lav belastning vil der tages emner fra den nedre grænse til kasser nær den øvre grænse. Ved høj belastning vil der ligeledes være lange diagonale bevægelser, men fra hvilken ende og til hvilken ende kommer an på situationen, før der kom høj belastning.



Figur 2.21: Her ses bevægelsesmønstret for LPT, som ved høj og lav belastning vil lave nogle lange diagonale bevægelser. Ved lav belastning vil der altid tages emner fra den nedre grænse, som pakkes i kasser ved den øvre grænse.

Modificeret SPT og Forbedret SPT er opbygget således, at de opererer ud fra et bestemt punkt hvor objektet skal placeres [Mattone et al., 1999, s.88-89]. I simuleringen vil den kasse, som er nærmest emnet, blive valgt - hvilket ofte vil være en forholdsvis central kasse. Derfor bør disse regler stadig virke efter hensigten.

Som forbedring til F-SPT, har projektgruppen udarbejdet en ny prioriteringsregel, kaldet Modificeret Forbedret SPT(MF-SPT). Idéen er, at gøre bredden af det optimale område i Forbedret SPT variabelt, istedet for at den konstant er $L = V * Tm$. Dvs. at konstanten Tm, som er den maksimale procestid, skal skiftes ud med en variabel tid, Tv. Når Tm, den maksimale procestid anvendes, giver det en meget sikker løsning, som vurderes at være overdimensioneret. Derfor vil ændringen i MF-SPT være at udregne Tv som procestiden for SPT objektet, der befinder sig i downstream området, altså procestiden for O_2. Dette vil for det meste medføre, at det optimale område bliver mindre, men målet med prioriteringsreglen vil stadig være opfyldt. Dette vil sandsynligvis betyde, at flere objekter i downstream området behandles, da flere emner befinder sig i upstream området, hvilket kan ses på figur 2.22 på næste side.



(a) Tm lig med maksimal procestid (b) Tv lig med variabel procestid

Figur 2.22: Her ses eksempler, hvor henholdsvis den maksimale procestid, Tm og den variable procestid, Tv, er anvendt. Det kan ses at ved Tm befinder der sig flere objekter i det optimale område, som skal behandles først. Modsat ved variabel Tv, vil det sandsynligvis være muligt at behandle objekt 1, som befinder sig i downstream området, og efterfølgende stadig nå objekt 2 inden det forlader det optimale område.

2.4.5 Pakkestrategi i simuleringen

Med de forskellige regler og begrænsninger kan der nu opsummeres, hvordan en pakkestrategi skal kunne sammensættes i simuleringen.

Heuristiske prioriteringsregler

For emner skal brugeren kunne vælge imellem:

- FIFO - First In First Out
- LIFO - Last In First Out
- SPT - Shortest Processing Time
- LPT - Longest Processing Time
- M-SPT - Modificeret SPT
- F-SPT - Forbedret SPT
- MF-SPT - Modificeret Forbedret SPT

Når én af disse regler vælges, vil det gælde for alle robotceller i anlægget. Kasserne vil køre med samme prioritering som emnerne - i tilfælde af M-SPT, F-SPT og MF-SPT skal kasserne køre med SPT.

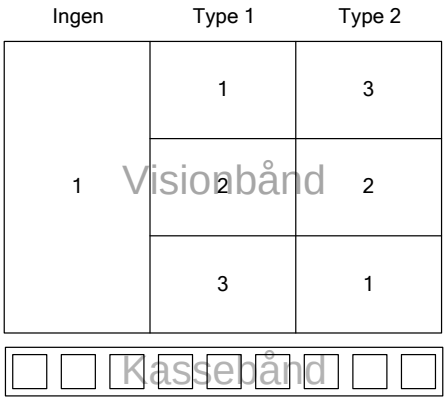
Det skal yderligere være muligt at vælge en advanceret opstilling, hvor man kan vælge en prioriteringsregel for hver enkelt celle - brugeren kan dermed køre med f.eks. SPT på robotcelle 1 og LIFO på robotcelle 4. Når brugeren vælger advanceret, skal der også være mulighed for at vælge prioritering af kasserne - der skal være følgende muligheder:

- FIFO - First In First Out
- LIFO - Last In First Out
- SPT - Shortest Processing Time

Der ses ingen logisk grund til, at brugeren ville vælge LPT til kasserne, den er derfor udeladt.

Vertikal prioritering

Brugeren skal her have mulighed for at kunne vælge mellem de tre typer vertikal prioritering for alle robotcellerne.



Figur 2.23: Med det lille kassebånd forneden og visionbåndet foroven, er der her illusteret, hvordan emnerne på visionbåndet kan prioriteres i den vertikale retning. Der er to typer: Type 1 - Her prioriteres emnerne længst væk fra kassebåndet. Type 2 - Her prioriteres emnerne tættest på kassebåndet.

Minimum antal targets

Brugeren skal have mulighed for at vælge det antal frie targets, som minimum skal være tilbage i kassen, når den forlader en robotcelles arbejdsområde. Der skal derfor

være mulighed for at vælge dette for hver robotcelle og for begge type kasser.

Bekræftigelse af pakkestrategi

For at få bekræftiget fremgangsmåden, hvorpå pakkestrategien skal sammensættes, er Peter Nielsen, adjunkt ved Institut for Mekanik og Produktion og forsker inden for produktionsplanlægning, blevet præsenteret for problemstillingen [Nielsen, 2011]. Han var enig i konklusionen om at problemet var for komplekst til at det kunne realtidsskeduleres, især hvis det skulle reskeduleres hver eneste gang, der kom nye emner ind i anlægget. Skulle der skeduleres, skulle det ikke reskeduleres så ofte - man skulle altså holde fast i en skedulering indtil et vis antal emner var blevet pakket. Dette ville dog stadig være en voldsom opgave, som Peter Nielsen mente at man sagtens kunne afsætte 2 Ph.D. studerende til at forske indenfor, for at en løsning kunne findes.

Han bekræftigede derfor idéen om at gå til opgaven med heuristiske prioriteringsregler, vertikal prioritering og minimum antal ledige targets. Man burde ud fra disse kunne udarbejde nogle kombinationer, som ville give en tilstrækkelig god performance.

Dette konkluderer hermed analysen.

3 Problemformulering

I analysen blev det klarlagt at skeduleringsmetoder, hvor der konstant søges efter en optimal løsning, ikke kan anvendes i pakkeanlægget. Dette skyldes især, at der er for kort tid til at skedulere arbejdsopgaverne og tildele disse til robotterne, inden nye arbejdsopgaver tilkommer. Desuden er der så mange variabler, forbehold og begrænsninger der skal tages højde for, at det kan være noget nær umuligt at finde optimum. Ifølge Peter Nielsen ville det kræve et par Ph.D. projekter at forske omkring og udarbejde en sådan skeduleringsalgoritme.

Det blev dermed bekræftet, at i et system så dynamisk som pakkeanlægget, vil løsningen mht. at fordele arbejdsopgaver være heuristiske prioriteringsregler kombineret med andre forbehold, som tilsammen kan danne en pakkestrategi. Derfor blev prioriteringsreglerne FIFO, LIFO, SPT, LPT, Forbedret SPT, Modificeret SPT og en variant heraf Modificeret Forbedret SPT beskrevet. Yderligere synes vertikal prioritering fordelagtig, da kan sikre hvilke robotter, der tager sig af de mest tidskrævende arbejdsopgaver. En sidste indstilling til en pakkestrategi er definering af et minimum antal frie pladser, som skal være tilbage i kasserne, når de forlader et arbejdsområde - dette kan bruges til fordeling af arbejdsbelastningen. Disse indstillinger skal være mulige i simuleringen og kombinationerne af disse vil give et stort antal pakkestrategier, som en bruger vil kunne afprøve i forsøget på at finde en strategi, der passer til et specifikt anlæg.

Til at udvikle software, blev der indhentet viden om udviklingsprocessen inden for softwareudvikling. Det blev klarlagt, at iterationsmodellen er procesmodellen, som skal anvendes til at udvikle simuleringen. Dette skyldes især, at en initierende simulering blev udviklet på 9. semester og udfra denne videreudvikles simuleringskonceptet. Det synes også fordelagtig at anvende denne model, da ikke alle specifikationer er bestemt, og derfor vil der løbende komme ændringer og forbedringer til simuleringen.

Yderligere blev der klarlagt nogle generelle specifikationer og krav til simuleringen, designregler mht. at udvikle en brugerflade blev undersøgt og der blev fundet metoder til at dokumentere software i form af UML sekvens diagrammer og pseudo kode.

På baggrund af analysen kan der udvikles en simulering af pakkeanlægget i C#. Denne simulering skal valideres, således at de programmerede komponenter, der indgår i simuleringen, er valide ifht. de tilsvarende fysiske komponenter i pakkeanlægget. Derefter vil der blive testet en række udvalgte pakkestrategier i forsøget på at maksimere/minimere de opstillede kriterier i tabel 1.2 på side 4.

Derfor opstilles følgende problemformulering:

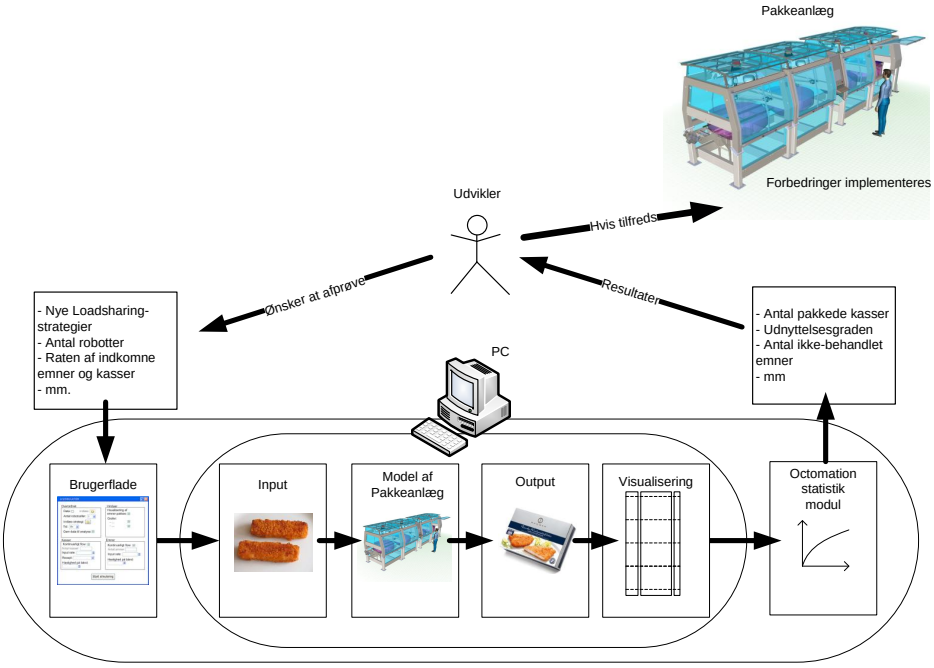
> *Der skal udvikles et fleksibelt og brugervenligt simuleringsværktøj, som kan anvendes til afprøvning og vurdering af pakkestrategier*

4 Simuleringssystemet

I dette kapitel gennemgåes opbygningen af simuleringssystemet, hvor der startes med at præsentere brugerfladen. Efterfølgende gives en detaljeret beskrivelse af simuleringssystemet, hvor strukturen i programkoden og en stepvis gennemgang af opbygningen præsenteres. Komponenterne som indgår i simuleringssystemet forklares, hvor der bl.a. anvendes UML sekvens diagrammer og Pseudo kode.

4.1 Formål

Som beskrevet tidligere er formålet med dette simuleringsprogram at bidrage til udviklingen af nye pakkestrategier samt generelt give et overblik over, hvordan belastningen af et anlæg kan variere ifht. input rate, antal robotceller, mm. Som det ses på figur 4.1 indgår simuleringen som et arbejdsredskab for en udvikler af pakkeanlæg. Når der er opstået nye idéer til styringen, eller en ny størrelse anlæg skal testes af, kan udvikleren bruge simuleringen til at fintune denne styring, indtil han finder outputtet tilfredsstillende. Da kan det implementeres i selve pakkeanlægget.



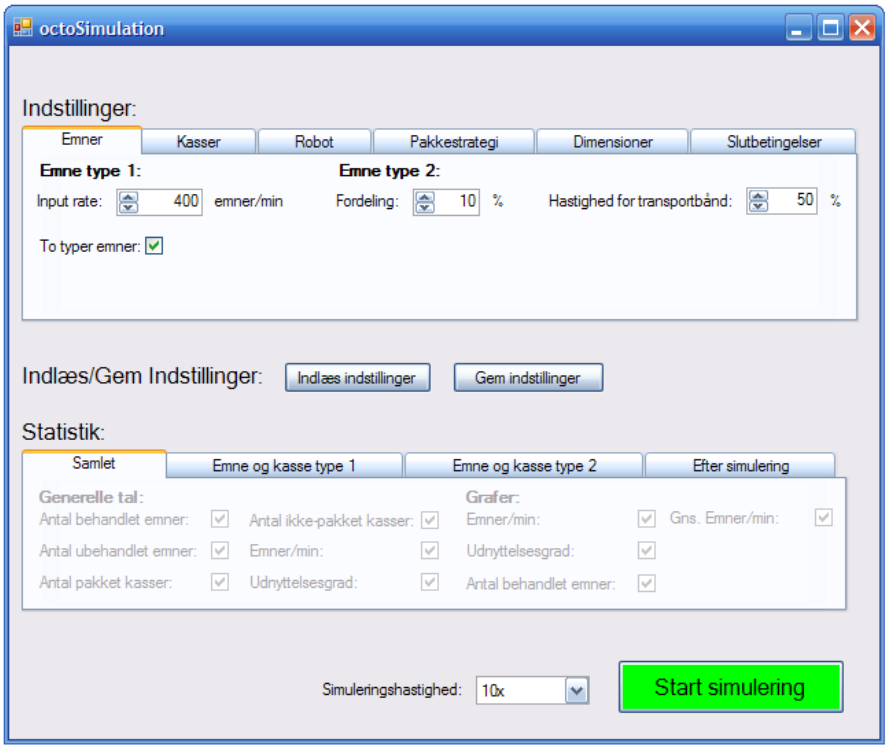
Figur 4.1: Simuleringen indgår i en arbejdsproces til udvikling af nye strategier og test af nye anlæg. Når udvikler er tilfreds med outputtet, kan en evt. ny strategi implementeres til virkelige pakkeanlæg.

4.2 Simuleringssystemets brugerflade

I dette afsnit vil simuleringssystemets brugerflade blive gennemgået. Brugerfladen består af en hovedmenu og et simuleringsvindue.

4.2.1 Hovedmenu

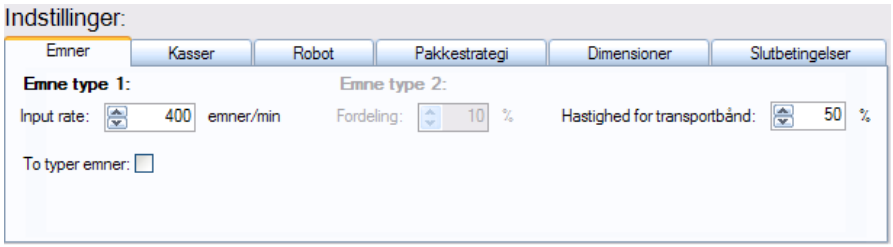
På figur 4.2 ses Hovedmenuen, hvor alle indstillinger til simuleringen opsættes. Hovedmenuen består af to faneblade hhv. Indstillinger og Statistik. Yderligere er det muligt at gemme eller indlæse indstillinger i XML format. Den sidste indstilling inden simuleringen startes, er hastigheden for simuleringen, som per default står til 10x - det er muligt at vælge 1x, 10x, 100x og 1000x. I det efterfølgende gennemgåes de enkelte faner i Indstillinger.



Figur 4.2: Hovedmenuen er delt ind i Indstillinger for simuleringen foroven og Statistik forneden.

Fanebladet Indstillinger består af fanerne Emner, Kasser, Robot, Pakkestrategi, Dimensioner og Slutbetingelser. På figur 4.3 på næste side ses fanen Emner, hvor indstillinger som input rate af emner og hastigheden for visiontransportbåndet sættes. Yderligere

er det muligt at tilvælge type 2 emner, altså 2. sorteringsemner, og indstille hvor stor en andel af input raten der skal være type 2 emner. Tilvælges to typer emner bliver der i fanen Kasser opstillet yderligere indstillinger, som er til kasse type 2, hvor type 2 emnerne skal pakkes i.



Figur 4.3: Under Emner tilvælges type 2 emner, samt input rate, fordelingsprocent og båndets hastighed.

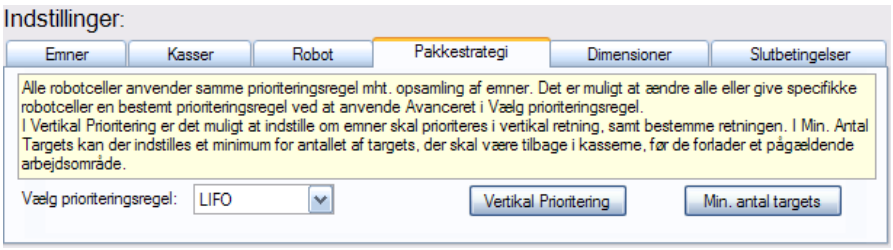
I fanebladet Kasser, som ses på figur 4.4, indstilles først antallet af targets, altså frie pladser, som skal være i hver kasse. Yderligere sættes input raten for kasser samt hastigheden for det tilhørende transportbånd. De samme indstillinger findes for Kasse type 2.

Figur 4.4: Under Kasser vælges antal targets, input rate og hastighed for båndene.

Under Robot fanebladet, som ses på figur 4.5 på næste side, sættes antallet af robot-celler, som kan være minimum 1 og maksimalt 8. Yderligere er det muligt at sætte robotternes acceleration, som standard står til 20 m/s^2. Det anbefales ikke at ændre på denne, da procestiderne ved denne opsætning ligger på 4-600 ms, som passer godt overens med virkeligheden.

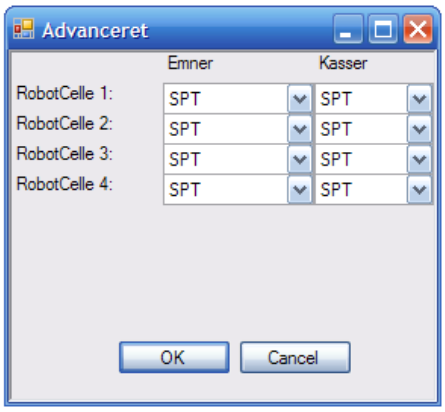
Figur 4.5: Under Robot kan vælges antal robotceller og robottens acceleration.

I fanebladet Pakkestrategi, som ses på figur 4.6, er det muligt at sætte prioriteringsregler, vertikal prioritering og minimum antal targets, som skal være ledige når en kasse forlader et arbejdsområde.



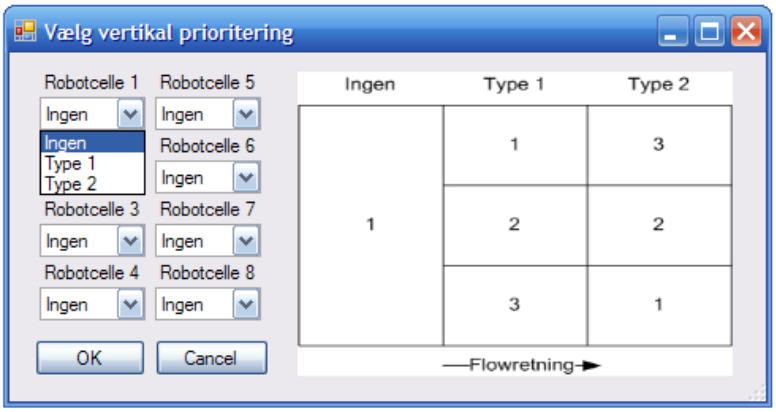
Figur 4.6: Under Pakkestrategi kan der vælges prioriteringsregler, vertikal prioritering og minimum antal targets

Som prioriteringsregel kan der vælges mellem LIFO, FIFO, SPT, LPT, M-SPT, F-SPT, MF-SPT og Advanceret. Hvis der vælges en prioriteringsregel gælder denne på alle robotceller. Hvis der ønskes at vælge en prioriteringsregel til hver enkelt robotcelle skal Advanceret vælges. Her åbnes et seperat vindue, som kan ses på figur 4.7 på næste side, hvor det er muligt at vælge prioriteringsregler til robotcellerne, både mht. emner og kasser.



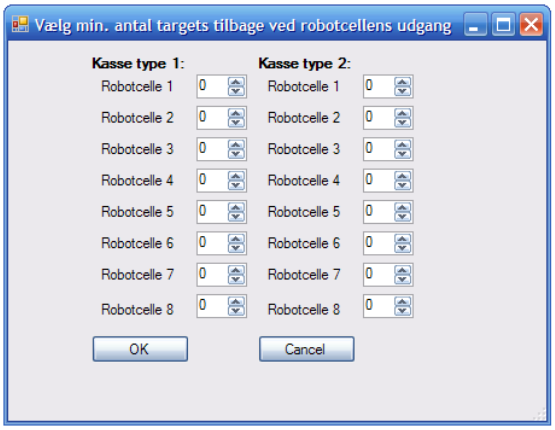
Figur 4.7: Her ses Advanceret, hvor der kan vælges prioriteringsregler i hver robotcelle. Der kan vælges prioriteringsregel til Emner og til Kasser.

Når man i Pakkestrategi vælger Vertikal prioritering, åbnes der et seperat vindue, som kan ses på figur 4.8. I vinduet kan der sættes en vertikal prioritering på hver enkelt robotcelle, type 1 eller 2, hvilket er beskrevet i afsnit 2.4.2 på side 27.



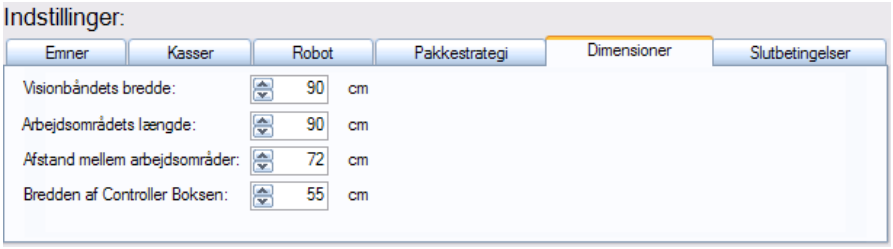
Figur 4.8: Her kan vælges vertikal prioritering for hver robotcelle.

Når man under Pakkestrategi vælger Min. Antal targets, åbner en nyt vindue, se figur 4.9 på modstående side. Her vælges det minimum antal af targets, som skal være tilbage når en kasse forlader et arbejdsområde. Der kan vælges for begge kasse typer.



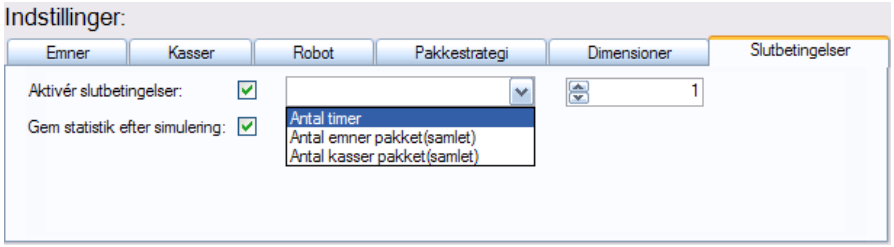
Figur 4.9: I dette vindue sættes der i hver robotcelle et minimum antal targets, som kasserne skal have inden de forlader det pågældende arbejdsområde.

I fanebladet Dimensioner er det muligt at ændre dimensionerne for visionbåndets bredde, arbejdsområdets længde, afstand mellem arbejdsområder og bredden af controller boksen.



Figur 4.10: Under Dimensioner kan enkelte dimensioner for anlægget indstilles.

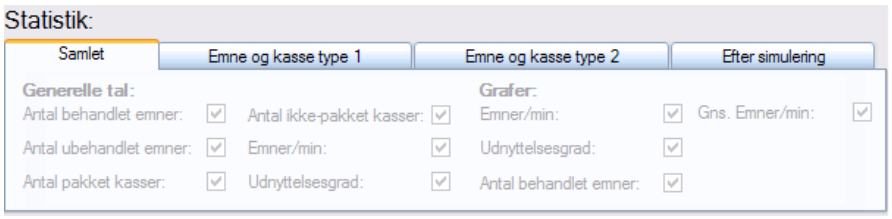
Den sidste fane under Indstillinger er Slutbetingelser. Ved at aktivere slutbetingelser, stopper simueringen når betingelsen er opfyldt. Betingelser, som kan vælges, er antal timer, antal emner pakket(samlet) og antal kasser pakket(samlet). Yderligere kan det vælges om der skal gemmes data til den efterfølgende statistik.



Figur 4.11: Brugeren kan her bestemme, om simuleringen skal slutte efter et hvis antal timer, pakket emner eller pakket kasser. Det kan yderligere vælges, om der skal gemmes data til den efterfølgende statistik.

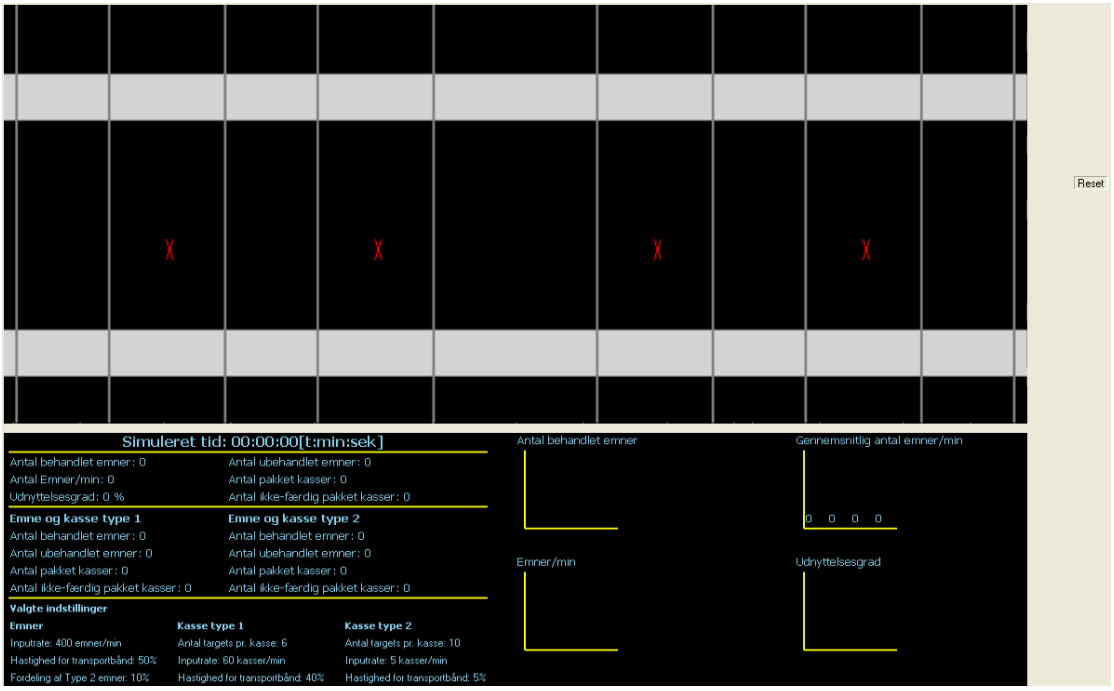
Statistik fanebladet består af følgende faner: Samlet, Emne og Kasse type 1, Emne og kasse type 2 og Efter simulering. Dette faneblad har været lavere prioriteret end resten af programmet, og blev ikke færdigudviklet indenfor projektets tidsramme. Det har derfor ingen funktion.

Formålet med fanebladet er imidlertidigt at kunne vælge hvilke grafer og tal, som der ønskes at se under simuleringen i Simuleringsvinduet. I fanebladet Efter simulering skulle det være muligt at vælge, hvad der ønskes vist i OCTOMATION Statistik modul. Dette modul blev aldrig tilkoblet, men i stedet er der udviklet et Statistik modul i MATLAB, se afsnit A på side 120, som kan analysere det gemte data - brugeren skal da huske at have tilvalgt Data til statistik under Slutbetingelser.



Figur 4.12: Fanebladet Statistik blev aldrig færdigudviklet, men skulle have givet brugeren mulighed for at tilvælge, hvilken statistik der ønskes under og efter simuleringen.

Hermed er Hovedmenuen og alle dets funktioner gennemgået. I afsnit 2.2.3 på side 11 blev en række designregler beskrevet, som er anvendt til at designe Hovedmenuen. Dette er bl.a. ikke at præsentere for meget information på en gang, hvilket er udført ved at lave fanebladene. Yderligere synes der at være en kontinuitet gennem Hovedmenuen, således brugeren ikke bliver overrasket i nogle af vinduerne.



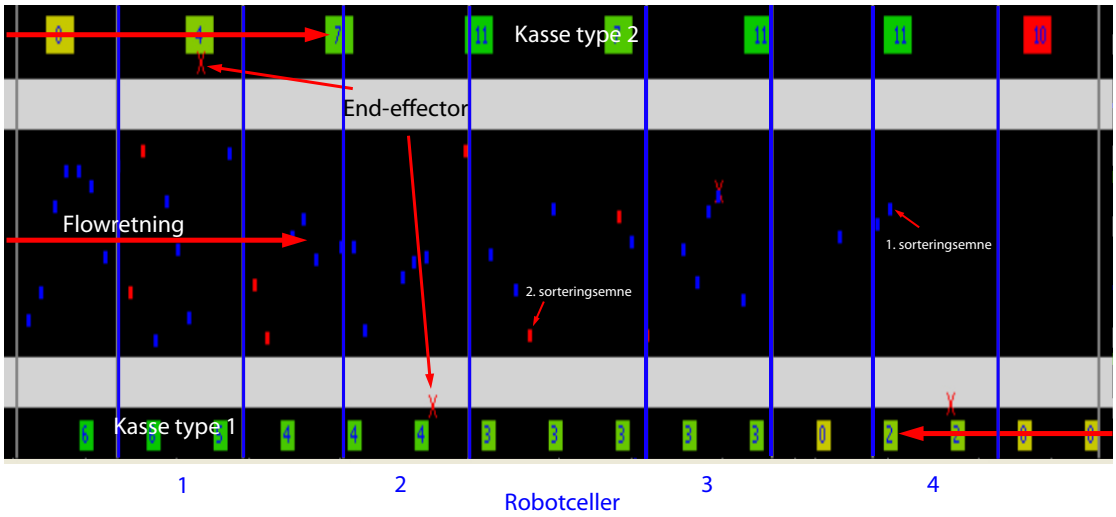
Figur 4.13: Simuleringsvinduet er opdelt i en visualisering øverst og en opremsning af nøgletal nederst.

4.2.2 Simuleringsvindue

Når simuleringen påbegyndes åbner et maksimeret vindue, som vist på figur 4.13. Vinduet er delt op i to - øverste del er en visualisering af emner og kasser, mens de kører hen ad de respektive transportbånd og bliver pakket. Den nederste del af simuleringen giver nogle nøgletal for hvordan anlægget performer, både samlet og for hver enkelt robot. Yderligere vises de valgte indstillinger i nederste venstre hjørne.

Visualisering

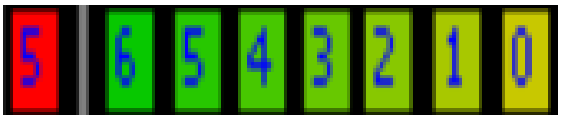
På figur 4.14 på den følgende side ses den øverste del af simuleringsvinduet, som er en grafisk simpel visualisering, som skal vise hvordan emnerne ligger, når de kommer ind i anlægget og bliver pakket af robotterne. Visualiseringen består af tre transportbånd - øverst er det store kassebånd med type 2 kasser, hvori 2. sorteringsemner skal pakkes. I midten ses visionbåndet, hvor emnerne ligger og nederst ses det lille kassebånd til type 1 kasserne, hvori 1. sorteringsemnerne skal pakkes. De store kasser og emnerne har begge to flowretning fra venstre mod højre, hvor de små kasser forneden har flowretning fra højre mod venstre. De blå emner er 1. sorteringsemner, de røde er 2.



Figur 4.14: Vinduet viser en visualisering af emner og kasser, mens de kører hen ad transportbåndene og bliver pakket

sorteringsemner. På figuren ses 4 robotceller, som hver har robottens End-effector vist med et stort rødt X.

Når først simuleringen er i gang, vil brugeren se End-effectoren bevæge sig hurtigt frem og tilbage mellem emnerne og kasserne. Emnerne vil blive pakket ned i de tilhørende kasser, så længe der er plads.

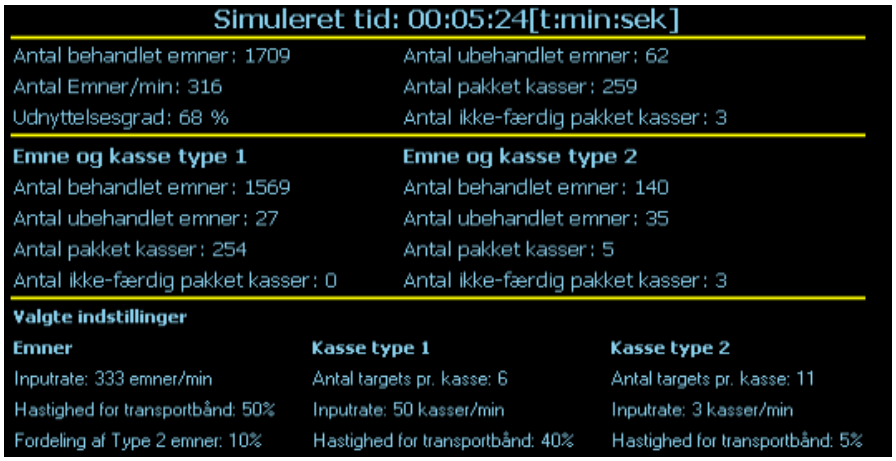


Figur 4.15: Farverne på kasserne går fra gul til grøn, efterhånden som de bliver fyldt. Når de til enden af anlægget, uden at være fyldt, vil de skifte til rød.

På hver kasse står et tal - dette viser hvor mange emner, der pt. ligger nede i kassen. Efterhånden som kassen fyldes op, vil den skifte farve fra gul til grøn, som vist på figur 4.15. Hvis en kasse når forbi den sidste robotcelle, uden at blive pakket, vil den skifte til en skarp rød farve.

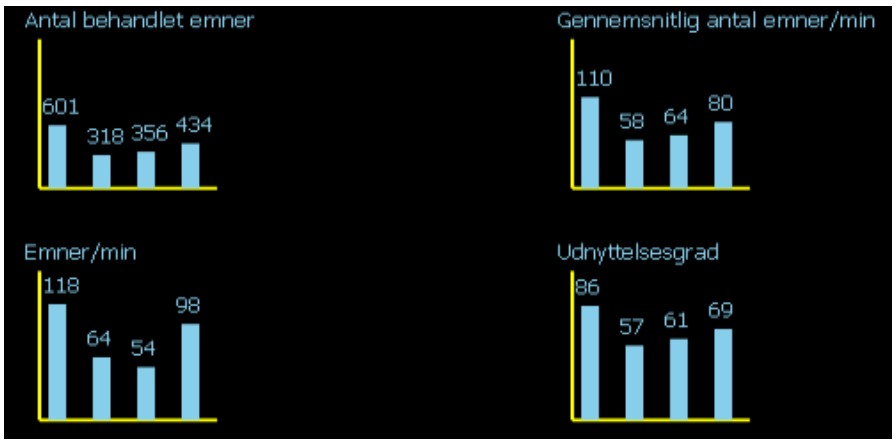
Nøgletal

Under simuleringen vil der blive vist nøgletal for, hvordan anlægget performer, se figur 4.16 på modstående side. Øverst på figuren ses den simulerede tid. Under den kommer samlede tal og gennemsnit for hele anlægget. I midten ses der tal for hver



Figur 4.16: Brugeren kan med disse nøgletal løbende under simuleringen holde øje med, hvordan anlægget performer. Det er yderligt muligt at se valgte indstillinger.

type emne og kasse, så man kan holde de to statistikker adskilt. Nederst vises de indstillinger, som brugeren har valgt ude i Hovedmenuen.



Figur 4.17: Søjlerne giver et billede af, hvordan hver enkelt robotcelle klarer sig ifht. de andre.

Figur 4.17 viser det nederste højre hjørne af simuleringsvinduet. Der bliver her vist nogle søjler over, hvordan hver enkelt robot performer mht. antal behandlede emner, det gennemsnitlige antal emner/min over hele simuleringstiden, antal emner pakket inden for det sidste minut og til sidst udnyttelsesgraden over hele simuleringstiden.

Med brugerfladen præsenteret vil den overordnet struktur blive gennemgået i næste afsnit.

4.3 Overordnet struktur

Figur 4.18 på modstående side viser den overordnede struktur af den opbyggede simulering.

Brugerflade

Når selve modellen af pakkeanlægget starter, vil de valgte indstillinger fra menuen blive læst ind.

Transportbåndene

Modelleringen af transportbåndene starter med deres *encoder* værdier, som løbende bliver opdateret. En *input* funktion genererer nye emner og kasser ifht. input raten og lægger disse ind på de tilhørende lister. Der findes tre type lister; én for emner, én for kasser på kassebånd 1 og én for kasser på kassebånd 2. Hver arbejdsområde på båndene har en liste. En *kontrol* funktion sørger for, at emner/kasser befinder sig på en korrekte liste ifht. hvilket arbejdsområde de befinder sig i - den kontrollerer hvor emner/kasser befinder sig ifht. encoderværdierne for båndene.

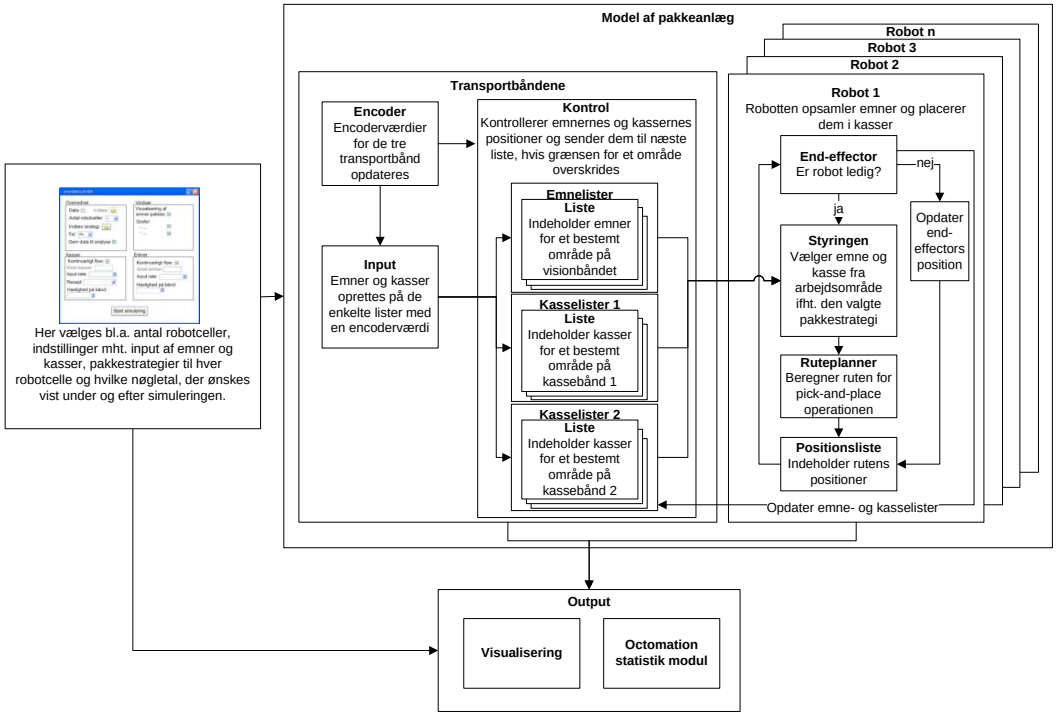
Robotterne

Hver robot kontroller om deres *End-effector* er ledig. Hvis det er tilfældet aktivere den *Loadsharing*, som finder et emne og kasse i robottens arbejdsområde, udfra listerne og vha. den valgte loadsharingsstrategi. Når emne og kasse er valgt, vil en *Ruteplanner* beregne ruten for pick-and-place operationen, hvilket gemmes i en *Positions liste*. Hver robot har sådan en liste - og når den ikke er ledig, opdatere den sin *End-effectors* position udfra denne liste.

Robotten vil desuden opdatere kasse- og emnelisterne, hver gang et emne tages op og placeres i en kasse.

Output

Hele modellen genererer et output, som indeholder data omkring emner, kasser og robotterne. Disse data bruges da til visualisering og statistik.



Figur 4.18: Overordnede struktur på simuleringen.

4.4 Stepvis gennemgang

Når selve modellen af pakkeanlægget startes op, kan det illustreres som i algoritme 2. Vinduet, som viser transportbånd, emner og kasser, åbnes; knapperne indsættes; og den grundlæggende baggrund tegnes op. Herefter indlæses de startværdier og indstillinger, som brugeren har tastet ind eller valgt i det forrige vindue, altså Hovedmenuen.

Alle globale variabler, værdier og lister startes op og initialiseres når *initial()* kaldes.

Til sidst initialiseres *Ticker()*, som er en timer-funktion. Den fungere således, at den kalder *Tick()*, vist i algoritme 3 på den følgende side, hvert 30. ms.

Algorithm 2 Opstart

1: Åben vindue, optegn knapper og grundlæggende design
2: Indlæs indstillinger og startværdier
3: Kalder Initial()
4: Initialiser *Ticker()*

Algoritme 3, *Tick()*, indeholder en hovedløkke bestående syv funktioner, efterfulgt af *Vizulise* - en metode, som optegner bånd, emner, kasser m.m., som vist på figur 4.14 på side 46.

I hovedløkken bliver alle de udregninger og sekvenser udført, som er nødvendige for at foretage en enkelt iteration i simuleringen. Antallet af iterationer udført for hvert 'tick' kan varieres vha. variablen *ProgramHast*, som brugeren kan skrue op og ned på i løbet af simuleringen. F.eks. kan *ProgramHast* være sat til 10, hvilket vil gøre at hovedløkken bliver udført 10 gange i løbet af 30 ms - visualiseringen udføres stadig kun den ene gang.

Det er på denne måde muligt at skrue op og ned for hastigheden, hvormed simuleringen bliver udført. Den øvre grænse for hvor mange iterationer, der kan foretages på 30 ms, er da afhængig af hvor kraftig en computer, simuleringen bliver kørt på.

For hver iteration øges tiden med 'deltaT' og konsekvenserne heraf bliver da regnet ud.

Algorithm 3 Tick

```
1: for (i=0, i<ProgramHast, i++) do
2:    Beregn Tid = Tid + ΔTid
3:    Beregn VBEncoder = VBEncoder + ΔVBEncoder
4:    Beregn KBEncoder = KBEncoder + ΔKBEncoder
5:    Kalder metoden Input()
6:    Kalder metoden Kontrol()
7:    Kalder metoden EndEffector()
8:    Kalder metoden dataLogger()
9: end for
10: Kalder metoden Vizulise()
```

Main-loopet består, kort forklaret, af følgende metoder:

- *tidstæller*

  Tiden øges med ΔT.

- *deltaEncoder*

  Encoder-værdierne for vision- og kassebånd bliver øget med båndenes hastighed ganget ΔT.

- *Input*

Input sørger for at indsætte nye emner og kasser i systemet. Input raten for emnerne varierer en smule, mens den for kasserne holdes fast. Når emnerne sættes ind, bestemmes deres position på båndet udfra en fordeling.

- *Kontrol*

  Metoden kontrollere hvilket arbejdsområde emner og kasser befinder sig i. Denne viden skal bruges, når det skal fastsættes, hvilke emner og kasser de forskellige robotter skal pakke.

- *EndEffector*

  Denne metode fordeler arbejdet til robotterne, udregner deres bevægelser og opdatere End-effectorenes positioner.

- *dataLogger*

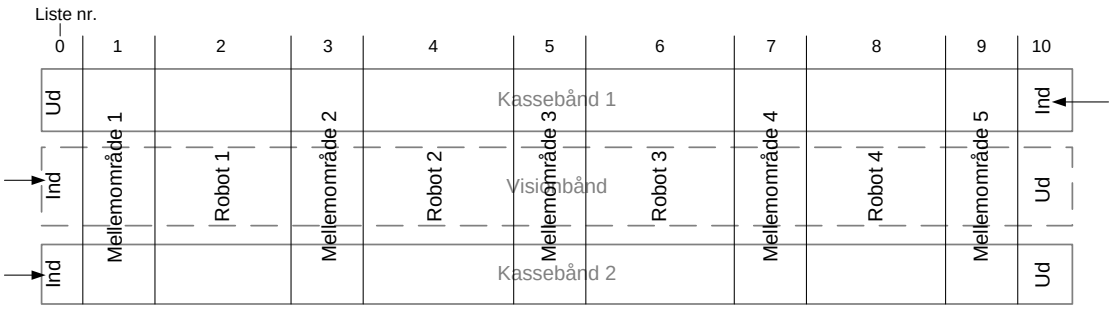
  Denne metode logger udnyttelsgraderne og emner pakket pr. min pr. robot - skal bruges til statistik efterfølgende.

Før metoderne gennemgåes i en detaljeret forklaring, er det vigtigt at forstå de grundlæggende elementer, som simuleringen er bygget op omkring. De vil derfor bliver forklaret i det følgende afsnit.

4.5 Liste-Systemet

Til at holde styr på alle emner og kasser i simuleringen bruges der et system af lister. Der findes tre forskellige lister, to til kassebåndene og én til visionbåndet. En liste repræsentere et bestemt område i pakkeanlægget og *Kontrol* sørger for at emner og kasser 'vandrer' korrekt op gennem listerne, efterhånden som de bevæger sig hen ad transportbåndene. Figur 4.19 på næste side viser hvordan Liste-systemet er blevet dannet til et pakkeanlæg med fire robotceller og to kassebånd. Listerne er nummeret således, at emne- og kasselisterne passer til hinanden. F.eks. repræsenterer emneliste 2 de emner, som kan pakkes ned i kasserne på kasseliste 2.

Som figuren viser er der i tilfælde af fire robotceller dannet 11 lister. Antallet af lister regnes ud som 2 gange antal robotceller(rc) plus 3. Det er to gange robotceller, da der for hver robotcelle er et arbejdsområde og et mellemområde til næste celle, som ikke kan nåes af nogle robotter. Udover dette vil der altid være brug for en liste til indgangen på båndene og en liste til udgangen på båndene. Den sidste liste er en "skraldespand",

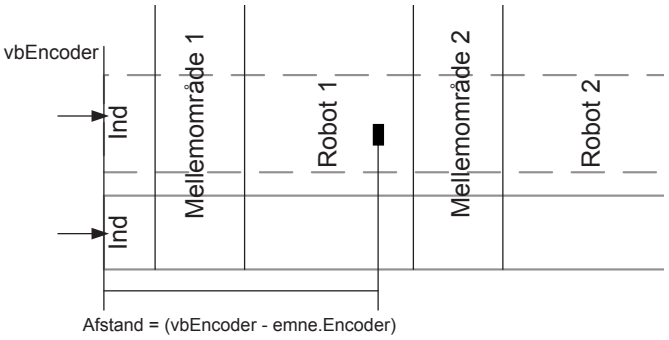


Figur 4.19: Visionbåndet og de to kassebånd er delt ind i de viste områder - for fire robotceller er der 11 inddelte rum, og dermed 11 lister. En indgang, en udgang, et arbejdsområde til hver robot og mellemområder mellem robotterne

når objekterne forlader pakkeanlægget. Dvs. for emnerne er "skraldespanden" en liste, hvor emner havner, når de ikke er blevet samlet op. For kasserne bliver det en liste over alle kasser, som har forladt anlægget og kan da bruges til at holde øje med, om alle kasser er blevet fyldt med emner.

4.6 Encoder

Til at holde styr på, hvor henne emner og kasser befinder sig i systemet, benyttes der en efterligning af det encoder system, som det virkelig anlæg bruger. Dvs. at båndene har en encoderværdi, som linært stiger for hver iteration(båndene kører med konstant hastighed) og når nye objekter kommer ind i systemet, tildeles de den daværende encoderværdi. Forskellen mellem objekternes encoderværdi og båndenes encoderværdi er da afstanden, som objekterne har bevæget sig, siden de kom ind i systemet. Se figur 4.20. Denne afstand bruges til at holde styr på, hvilken liste objekterne skal befinde



Figur 4.20: Forskellen mellem emnet og visionbåndets encoderværdi bestemmer emnets position

sig i. Den bruges samtidig også, når der skal regnes ud, hvor langt End-effectoren skal bevæge sig.

4.7 Forklaring af Main-loop

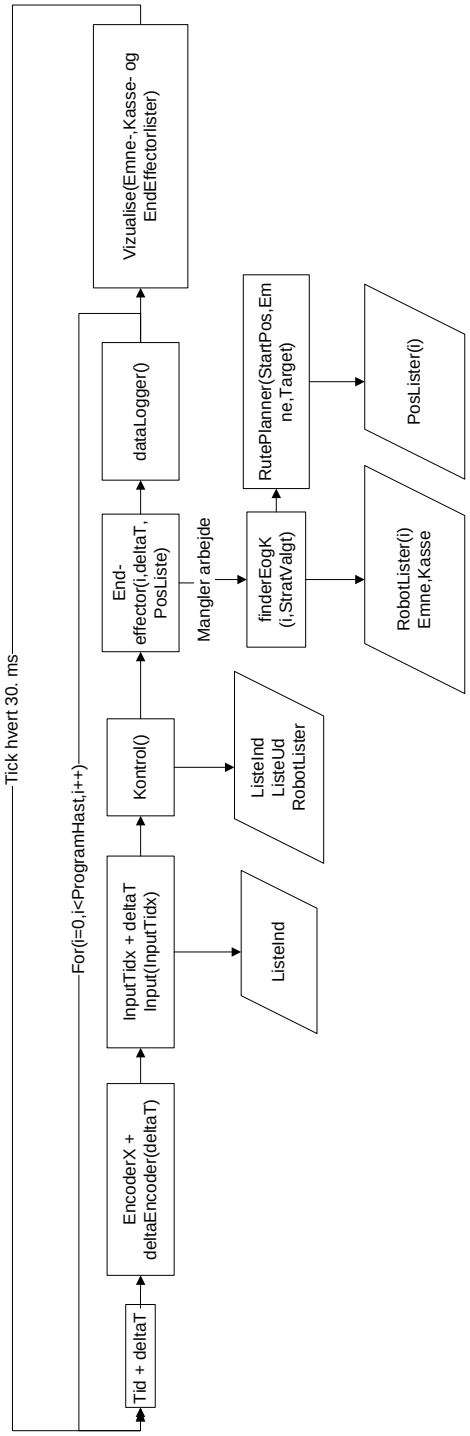
Der vil i dette afsnit blive gennemgået alle de forskellige funktioner og metoder, som Main-loopet består af. Først vil der komme en gennemgang af, hvordan Main-loopet fungere. Se figur 4.21 på side 55.

En iteration gennem Main foregår således:

- Tiden tæller ΔT op. Δ er på 20 ms.
- Båndenes encoderværdier adderes med deltaEncoder ganget deltaT, altså det stykke båndene har bevæget sig i tiden deltaT.
- Input() har en værdi kaldet InputTid til både kasser og emner. Hver gang et nyt objekt sættes ind på indgangslisterne, sættes denne tid lig nul. Den adderes efterfølgende i hver iteration med deltaT og når den opnår en bestemt størrelse, indsættes der et nyt objekt. Størrelsen, der skal opnåes, er den pågældende input rate, som er bestemt før simuleringen gik igang. input raten for emnerne har en vis variation.
- Kontrol() kontrollere første emnelisterne og da kasselisterne igennem. Den sørger for, at emner og kasser befinder sig på de rigtige lister i forhold til, hvilket arbejdsområde de befinder sig i.
- EndEffector() går robotcellerne igennem én efter én. Mangler en robotcelle arbejde, kontrolleres det om der er et emne og en kasse ledig i arbejdsområdet. Er det tilfældet, vil metoden finderEogK() finde et emne og en kasse til pick-and-place operationen, udfra den forudvalgte strategi for robotcellen. Emne- og Kasselisterne opdateres og emne og kasse returneres. Emne, kasse og End-effectorens startposition bruges nu i RutePlanner(), som planlægger den rute til End-effectoren. Denne rute deles op i steps, således at ét step svarer til, hvor meget End-effectoren bevæger sig i løbet af deltaT. Ruten gemmes i PosLister, som indeholder en liste for hver robotcelle. Næste gang EndEffector() kører vil den pågældende End-effector opdatere sin position i forhold til PosLister.

- dataLogger() udregner emner/min og udnyttelsesgraden for hver robotcelle og udskriver dette data i en .txt fil. Den begrænses til at sample for hver 50. iteration, hvilket med ΔT på 20 ms, giver en samplingsrate på 1000 ms eller 1 sekund.
- Vizulise() opdatere visualiseringen af kasser, emner og End-effectorene ifht. deres nuværende position.

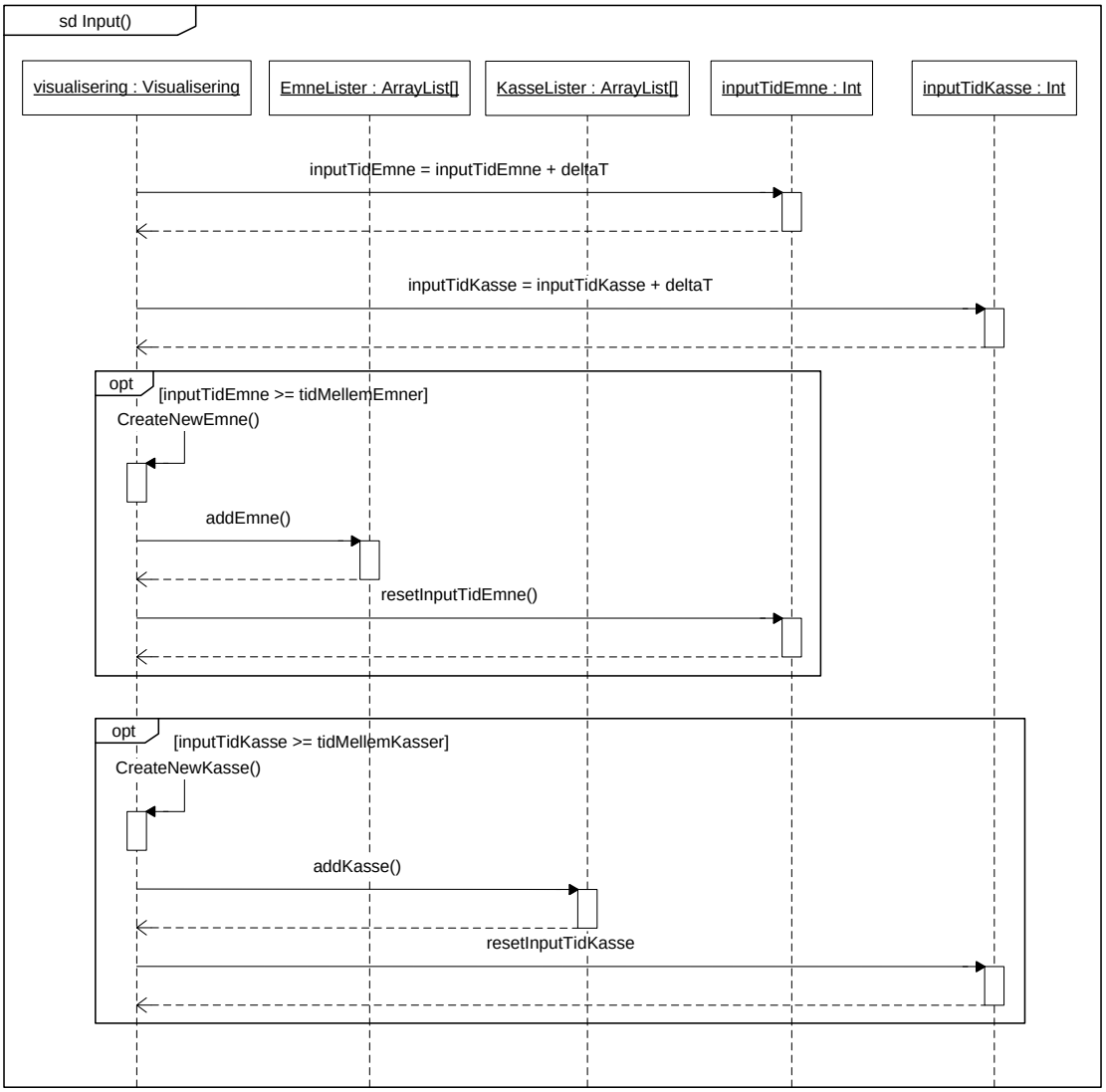
Hver iteration bliver kørt hvert 30. ms, da dette interval er det korteste interval, som stadig giver en tilfredsstillende visualisering. Med gennemgangen af en iteration på plads, vil der i de efterfølgende afsnit blive gået i dybden med at forklare en række af metoderne.



Figur 4.21: Main-loopet i en simpel optegning. Rektangler er metoder, romber er data

4.8 Input

Input() starter med at addere *deltaT* til både *inputTidEmne* og *inputTidKasse*. Derefter kontrolleres det om de to værdier har oversteget henholdsvis *tidMellemEmner* eller *tidMellemKasser*. Er dette tilfældet bliver der dannet enten nyt emne eller ny kasse, som lægges i indgangslisten ved enten Emne- eller KasseLister. Derefter nulstilles *inputTiden* og ved emnerne regnes en ny *tidMellemEmner* ud ifht. input raten og dennes variation. Når der skabes et nyt emne, kan det enten blive et 1. eller 2. sorteringsemne - dette bliver afgjort vha. fordelingsraten, som brugeren har valgt og en random-funktion. Det er på denne måde tilfældigt, hvornår 2. sorteringsemnerne foregår, men de forekommer stadig med den korrekte fordeling. Den nederste opt-boks, hvor der skabes og indsættes en ny kasse, forekommer i selve koden to gange - én for type 1 kasser og én for type 2 kasser. Figuren er her forsimplet.

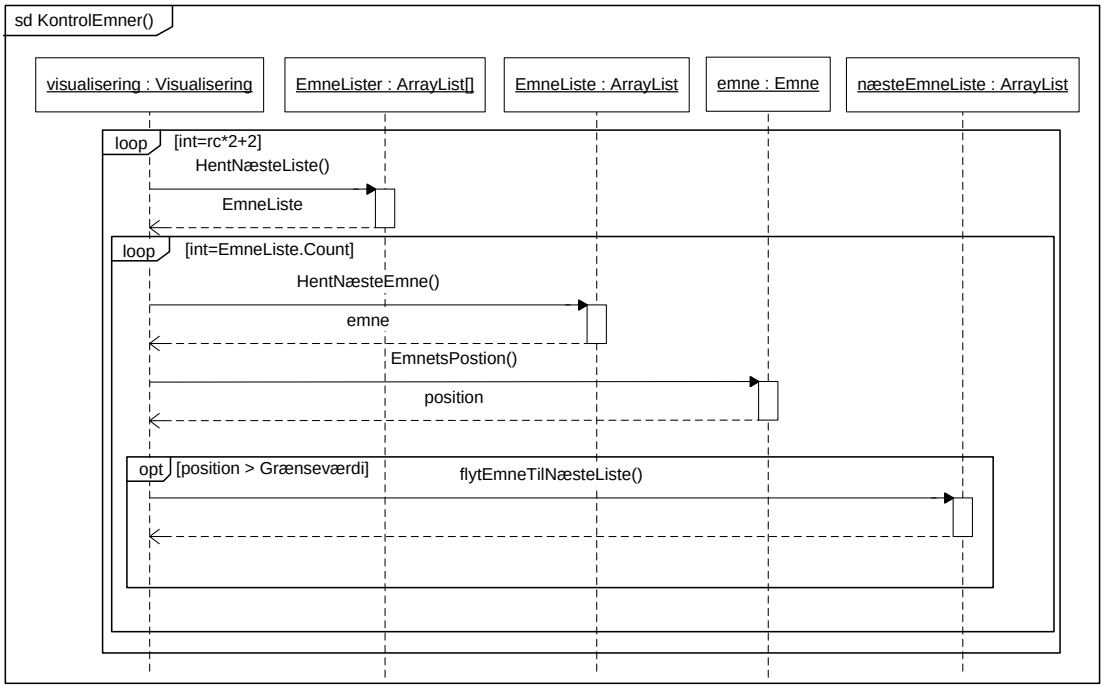


Figur 4.22

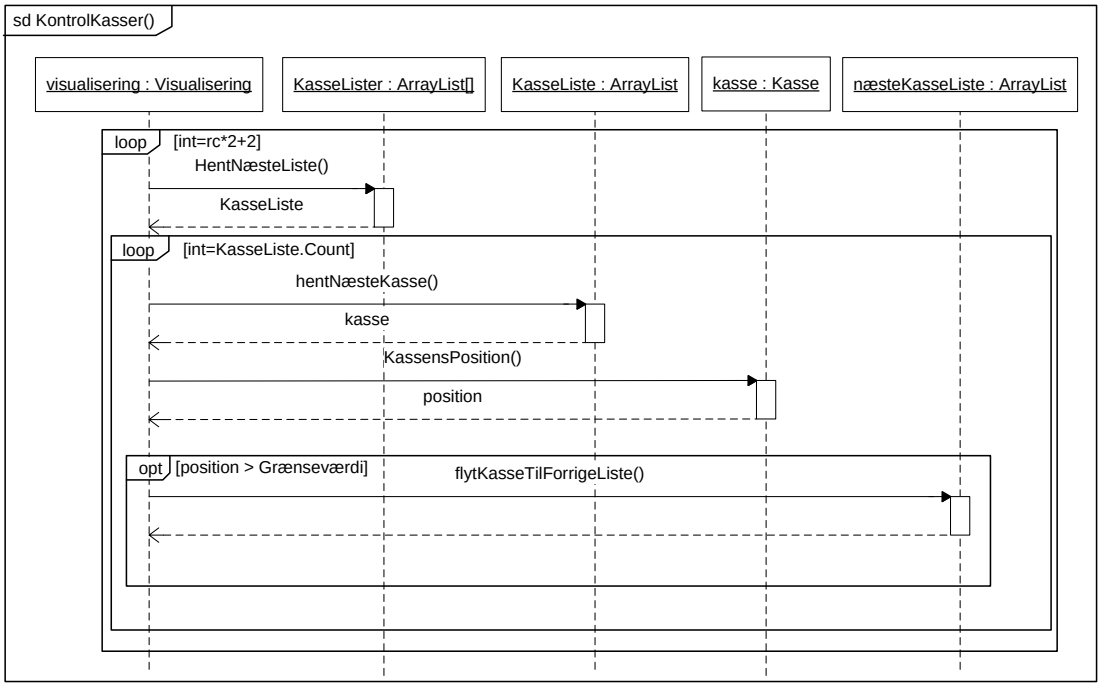
4.9 Kontrol

Kontrol() kan deles op i to sektioner, hvis eneste forskel er, om der kontrolleres efter emner eller kasser. Funktionen kører *rc**2+2 gange, dvs. alle lister undtagen den sidste kontrolleres. Se evt. afsnit 4.5 på side 51. Ved hver liste kører der en løkke lige så mange gange, som der er emner/kasser i listen. Hver emnes/kasses position bliver kontrolleret og overstiger den nogle forudsatte grænseværdier, flyttes den videre til

næste liste. Grænseværdierne repræsentere de forskellige arbejdsområders grænser.



Figur 4.23



Figur 4.24

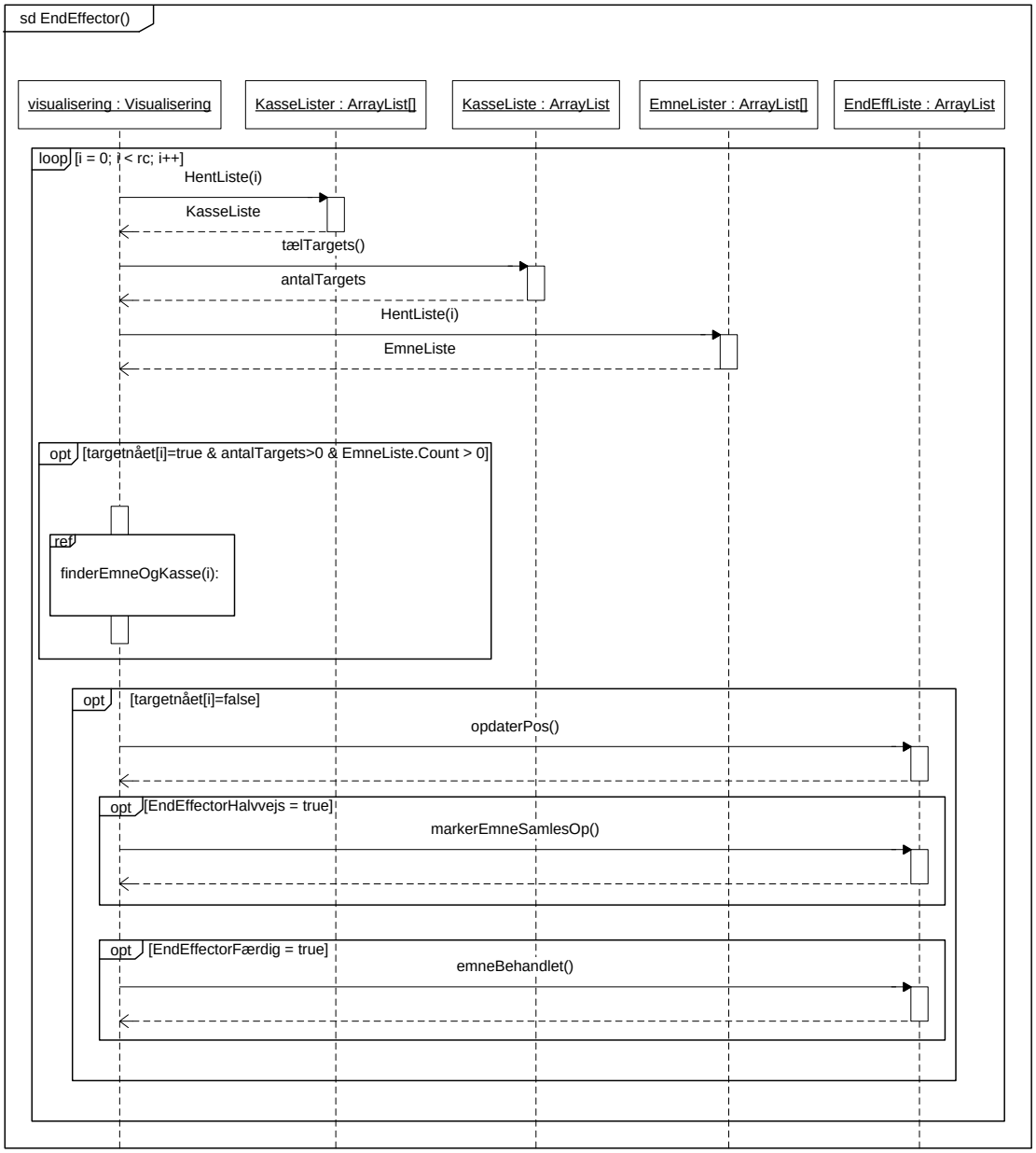
4.10 EndEffector

EndEffector() kører lige så mange gange, som der er robotceller(rc), indikeret af forløkken. Først kører *HentListe(i)*, som henter kasselisten for den pågældende robotcelles arbejdsområde. Derefter tælles alle ledige targets i dette område og gemmes i *antalTargets*. Da hentes emnelisten for det pågældende arbejdsområde.

Der er nu to muligheder - enten er End-effectoren igang med en pick-and-place operation eller ledig. Der er oprettet en boolværdi, *targetnået*, til hver End-effector. Hvis denne er *true*, er End-effectoren ledig. Er der samtidig targets og emner ledige i området (*antalTargets* > 0 & EmneListe.Count > 0), vil metoden *finderEmneOgKasse()* blive aktiveret. Denne finder et emne og target til End-effectoren, så den nu har en pick-and-place operation er gå igang, se afsnit 4.11 på side 61.

Er *targetnået* derimod *false*, er End-effectoren altså igang med en pick-and-place operation. End-effectoren rykker sig et step og dens position opdateres herefter - den følger en rute, *PosListe*, som er blevet udarbejdet i *finderEmneOgKasse*, da denne fandt ledig emne og target. To kontroller finder da sted; er End-effectoren halvvejs eller færdig.

Er End-effectoren halvvejs på *PosListen*, er den over emnet, hvilket markeres. I praksis har hver End-effector en boolværdi, *harEmne*, som her bliver sat *true*, hvilket visualiseringen bruger, når End-effectoren tegnes op. Er End-effectoren færdig bliver en række værdier ændret, for at markere dette. Bl.a. bliver *harEmne* sat til *false* og *targetnået* sat til *true*.



Figur 4.25

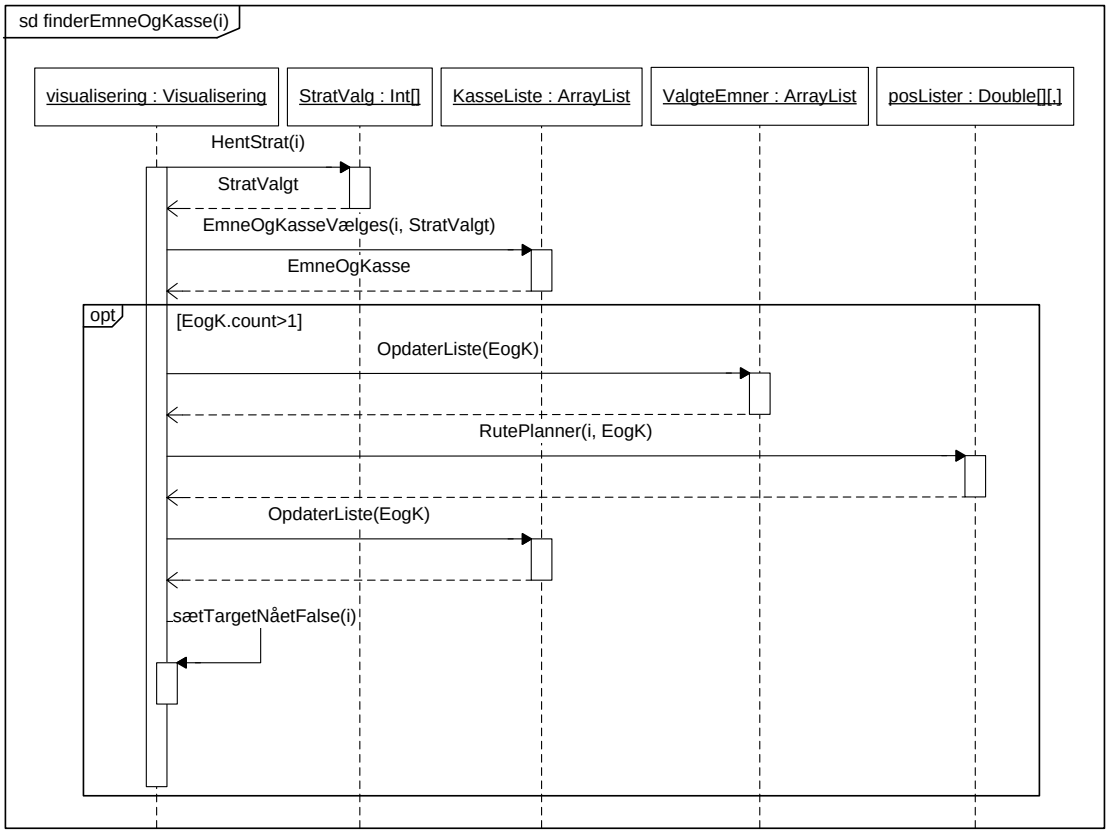
4.11 finderEmneOgKasse

Formålet med *finderEmneOgKasse* er at finde det næste match, dvs. emne og kasse, som robotten skal udføre en pick-and-place operation med. Dette er afhængig af hvilket

strategi, som er blevet valgt. Derfor bliver den valgte strategi hentet som det første. Dette bliver gjort ud fra en liste, *StratValg*, som dækker over hvilke strategier, som brugeren har valgt til hver enkelt robotcelle. Da køres *EmneOgKasseVælges(i,StratValgt)* som returner *EmneOgKasse* - det er altså under denne metode, at den reelle styring ligger, da det her bliver bestemt, hvilke emner og kasser der skal pakkes.

Det bliver herefter kontrolleret om *EogK*(EmneOgKasse) er større end én - dvs. indeholder to objekter, et emne og en kasse. Er dette tilfældet, bliver det valgte emne gemt i en arrayliste, *ValgteEmner*, der bliver brugt i andre metoder til at holde styr på, hvilke emner der behandles lige nu.

Ruten, som End-effectoren skal køre, for at samle emnet op og lægge i kassen, bliver da regnet ud og gemt af *RutePlanner*. Den gemte rute lægges i listen *posLister*. Derefter opdateres to lister; emnet fjernes fra den pågældende EmneListe og kassen bliver opdateret i KasseListen, så den har et ledig target mindre. Til sidst sættes *targetnået* til *false*, som forklaret i afsnit 4.10 på side 59.



Figur 4.26

4.12 RutePlanner

RutePlanner er en metode, som repræsenterer den del af robotstyringen, som udregner de kartetiske bevægelser for robotten, så den kan udføre en pick-and-place operation. Den udregner ruten, så End-effectoren rammer emnet/kassen på det korrekte tidspunkt og interpolerer ruten i steps ifht. deltaT. Denne interpoleret ruteplan kan nu bruges af *EndEffector*, når den skal opdatere dennes position.

Ved at observer figur 4.27 på næste side kan man se et sekvensdiagram af metoden. Når *RutePlanner* aktiveres, skal der bruges to input: Robotcelle nummeret, *celleNr*, og *EogK*, som er en liste der indeholder emnet og kassen, som skal pakkes. Ruten bliver planlagt ved først at regne ruten fra End-effectorens position til emnet, og derefter ruten fra emnet til kassen. De to ruter bliver til sidst lagt sammen til én samlet rute.

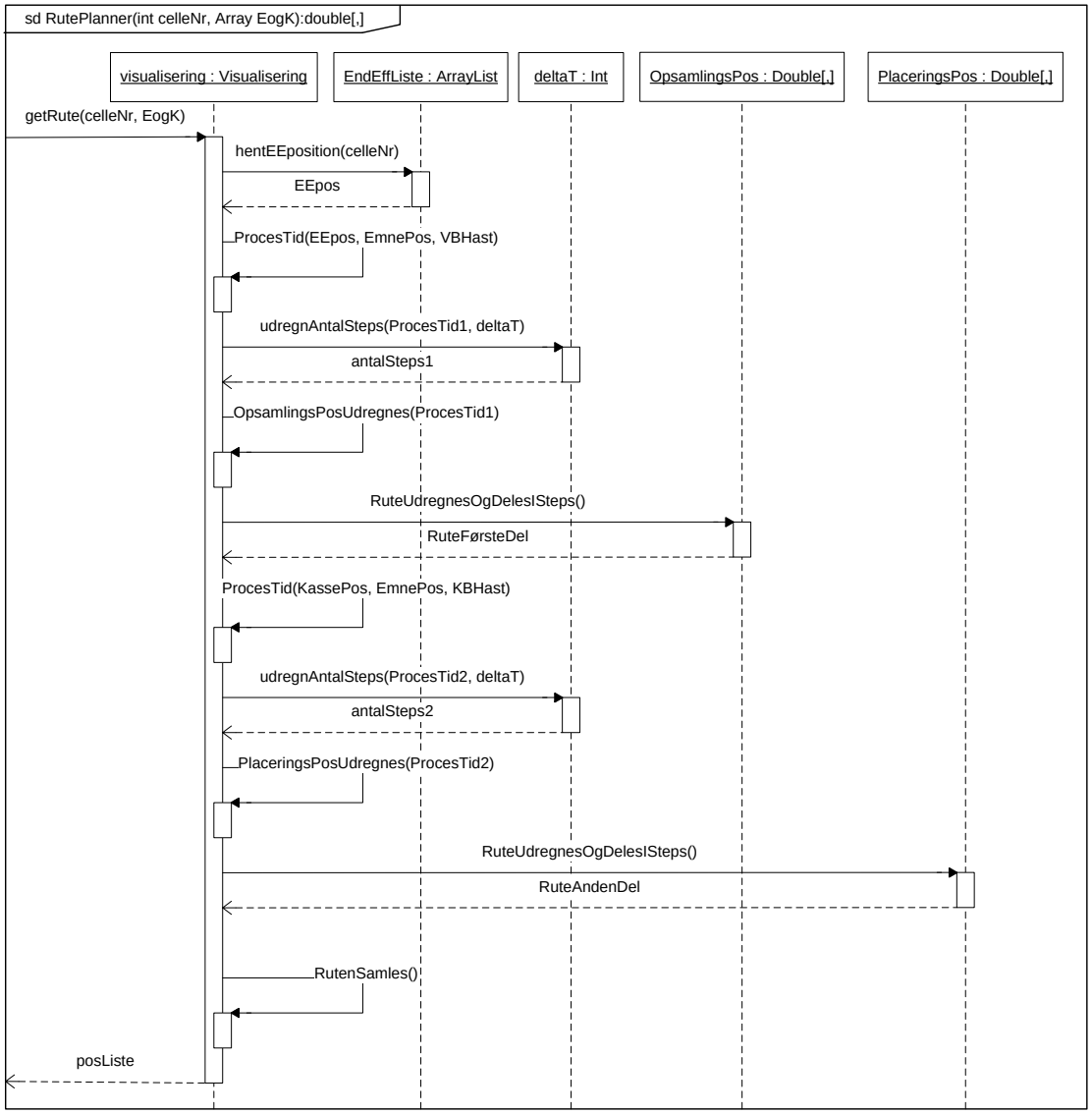
Først findes den pågældende End-effectors position; vha. *celleNr* findes den korrekte End-effector i en ArrayListe, *EndEffListe*.

Da bruges metoden *ProcesTid* til at regne ud, hvor lang tid det vil tage End-effectoren at køre hen og samle emnet op. Dette gøres vha. End-effectorens position, *EEpos*, emnets position, *EmnePos*, samt VisionBåndets hastighed, *VBHast*.

Vha. den udregnede procestid, *ProcesTid1*, og *deltaT*, udregnes antallet af steps, som den første del af ruten skal deles i.

Opsamlingsposition for emnet bliver da regnet ud vha. *ProcesTid1* og den første del af ruten kan nu udregnes og deles i det antal steps, som blev udregnet tidligere.

En næsten identisk proces udregner da den anden del af ruten, som går fra emnet til kassen - den bliver ligeledes interpoleret og gemt. Til sidst samles de to ruter til én samlet rute, som bliver gemt og returneret i en liste, *posListe*. Med en *posListe* for hver End-effector, ved disse altid hvor de skal kører hen.



Figur 4.27

4.13 ProcesTider

Ved udregning af procestiden antages det, at End-effectoren bevæger sig linært med en bestemt acceleration. Metoden ser ud som algoritme 4 på modstående side.

Den forudbestemte acceleration hentes, og en initerende procestid sættes. Der bliver da regnet ud, hvor langt der er mellem udgangspunktet og slutpunktet, når båndende har

Algorithm 4 ProcesTid(pos1, pos2, BåndHast)

```
1: HentAcc()
2: SætInitierendeTid()
3: 10:
4: DistMellem2PosEfterTid()
5: DistEEkørtEfterTid()
6: UdregnForskel()
7: if (forskel>e) then
8:     JusterTid()
9:     GOTO 10
10: end if
11: return Tid
```

kørt i den pågældende procestid. Derefter regnes der ud, hvor langt End-effectoren kan køre på den tilsvarende tid. Dette giver to distancer, og hvis fejlen/forskellen mellem disse er for stor, justeres tiden med en optimerings-algoritme, og programmet starter forfra (GOTO 10).

Optimerings-algoritmen brugt er en *steepest descent optimizer* [Endelt, 2009].

Først når fejlen/forskellen kommer under acceptabelt niveau, returneres den udregnede Tid. Forsøg viser, at den pågældende optimerings-algoritme finder frem til den korrekte tid på 6-8 iterationer.

Validering af simuleringssystemet

5

Som beskrevet i afsnit 2.2 på side 6 består en valideringsproces både af en verificering og en validering. I verificeringen kontrolleres det, at programmet lever op til de opstillede specifikationskrav opstillet i samme afsnit. Selve valideringen består af tre stadier:

- Test af komponenter
- Test af hele programmet
- Slut test

En slut test skal udføres med data fra kunden og i samarbejde med denne. Da det ikke er lykkedes at skaffe data fra OCTOMATION og der hellere ikke er fundet tid til at afprøve det endelig program i samarbejde med dem, så er denne sluttest udeladt af valideringen.

5.1 Verificering

Kravene opstillet i tabel 5.1 er dem, som blev opstillet i afsnit 2.2 på side 6. De er ikke så detaljerige, men beskriver alligevel meget godt, hvad forventningerne var til simuleringssystemet fra OCTOMATIONs side.

Simuleringssystem	
Funktion	Afprøve forskellige indstillinger i simuleringssystemet og efterfølgende udregne konsekvensen af indstillingerne mht. ydelsen for pakkeanlægget
Beskrivelse	I simuleringssystemt skal de være muligt at opsætte nogle ønskede indstillinger for pakkeanlægget, som efterfølgende skal simuleres. Nogle af de vigtigste indstillinger er: Antal robotceller; input rate for emner og kasser; Prioriteringen, som skal anvendes i de enkelte robotceller; Antal targets i kasser; Tilvalg af 2. sorteringsemner; Transportbåndenes hastighed; Efterfølgende skal det tydeliggøres hvorledes pakkeanlægget performede.
Input og source	Simuleringen bruger ingen datainput - alt genereres ud fra indstillingerne.
Output og destination	Pakkeanlæggets performance i form af f.eks. antal pakket og ubehandlet emner, antal færdig/ikke-færdig pakkede kasser, udnyttelsesgrad m.m., skal gemmes. Dette output skal anvendes i OCTOMATIONs statistik modul, som analyserer disse output og præsenterer pakkeanlæggets performance.
Aktion	Overordnet set skal simuleringssystemet generere emner og kasser udfra input rate m.m. Disse kører gennem pakkeanlægget på transportbåndene efter valgt hastighed på båndene. Hvis der er ledige emner og kasser i samme robotcelle, og robotten er ledig, skal emnerne pakkes i kasserne, efter gældende prioriteringsregler m.m.
Krav	-
For- og efterbetingelse	-
Sidevirkninger	-

Tabel 5.1: Simuleringssystemets systemkrav

Kravene er mere eller mindre alle blevet opfyldt. Det eneste sted, som har mangler, er planen om, at koble OCTOMATIONs eget statistik modul til simuleringen. Da det inden for projektets tidsramme aldrig lykkedes at anskaffe dette statistik modul, er dette ikke blevet til noget. Dog burde det ikke være svært, at koble et statistik modul til i fremtiden, da simuleringen pt. udskriver data, simpelt opstillet i .txt filer.

5.2 Validering

Som nævnt tidligere i afsnittet vil valideringen bestå af en test af komponenter og en test af hele programmet.

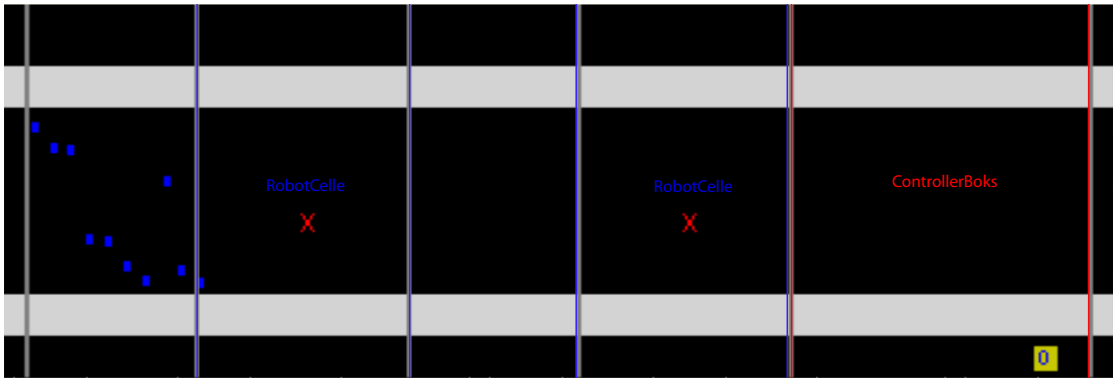
5.2.1 Test af komponenter

De forskellige komponenter vil blive testet i forsøget på at validere, om de fungere uden fejl og at de fungere korrekt som en simulering af deres fysiske pendant.

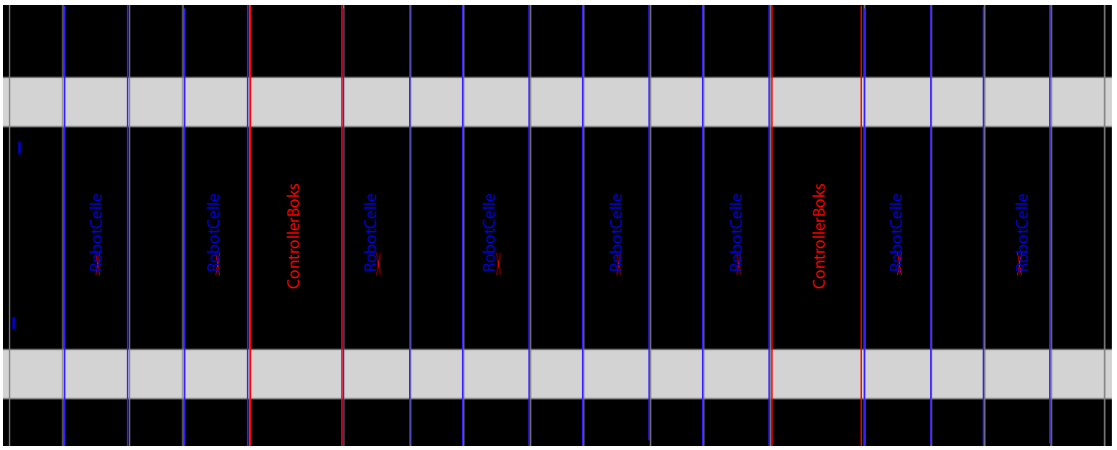
Dimensioner

For så vidt muligt er dimensioner oplistet i afsnit 2.1 på side 5 blevet brugt i selve simuleringen. Det er yderligere blev gjort muligt at ændre på visse dimensioner i indstillingerne - det drejer sig om visionbåndets bredde, arbejdsområdets længe, afstand mellem arbejdsområder og bredden af controller boksen.

Controller boksen giver et ekstra mellemområde midt på båndet, hvilket er vigtigt at tage højde for. Placeringen af controller boksen ændrer sig dog ifht. hvor mange robotceller anlægget består af [InMóTx, 2010]. Betragt figur 5.1 og 5.2 på modstående side. Her ses hvordan simuleringen tager højde for placeringen af controllerboksen alt efter, hvor mange robotceller anlægget består af.



Figur 5.1: Med kun to robotceller, placeres controllerboksen helt til højre i anlægget



Figur 5.2: Med otte robotceller, placeres de to controllerbokse med to robotceller på hver side, ligesom hvis der var fire celler

Transportbånd

Som forklaret i afsnit 4.4 på side 49 er transportbåndene i programmet simuleret ved, at de har en Encoder-værdi, som bliver konstant forøget med en bestemt værdi, Δencoder, gennem hele simuleringen. Δencoderen er bestemt ud fra den hastighed, som brugeren har valgt i starten af programmet. Hastigheden på båndene forbliver altså den samme gennem hele simuleringen og kan ikke ændres. Når et emner kommer ind i anlægget, tildeles det samme Encoder værdi, som transportbåndet pt. har, og forskellen på emnets og båndets encoderværdi bruges dermed som beregning for, hvor emnet befinder sig.

Dette er den samme måde, som det fysiske anlæg udregner emnernes positioner på båndet, så de passer godt sammen. Ligeledes bliver der aldrig ændret på visionbåndets hastighed, når anlægget kører. Derimod ændres kassebåndets hastighed sig faktisk i løbet af produktionen - en regulering sørger for, at kassebåndets hastighed og ankomsten af nye kasser passer sammen med belastningen af anlægget. Dvs. der skrues op og ned alt efter hvor mange emner der kommer ind i anlægget. Denne regulering mangler i simuleringen - dog er den ikke nødvendig, da belastningen af anlægget overordnet set ikke vil varierer i løbet af en produktion. Selvom der er en lille variation ifht. hvor og hvornår der ankommer emner i simuleringen, vil inputtet stadig overordnet forblive på det antal emner/min, man har bestemt i indstillingerne.

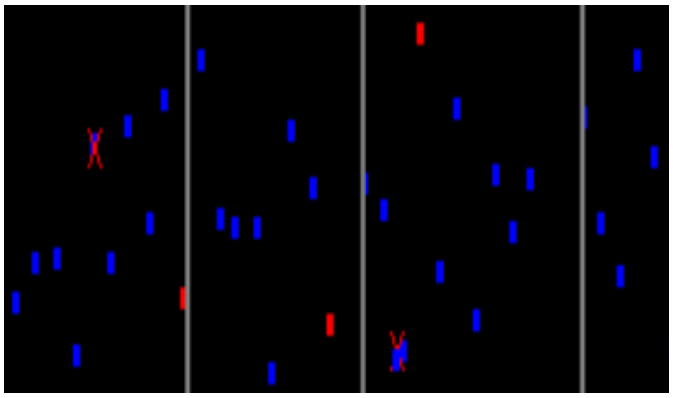
Emner

Når emnerne genereres i simuleringen bliver de tildelt værdier for, hvor i bredden af båndet de ligger, encoderværdien for båndet samt en bool-værdi for, om det er et 2. sorteringsemne. Hvor ofte der ankommer et nyt emne, bliver styret med en variabel, som er udregnet på baggrund af hvad brugeren har valgt i emner/min. Positionen i bredden af båndet bestemmes, så den naturligvis holder sig inden for rammerne af, hvor bredt båndet er blevet bestemt til at være af brugeren. Til at bestemme om det er et 2. sorteringsemne, bruges den fordelingsprocent, som brugeren har valgt.

Disse indstillinger har så dannet baggrund for en udregning, som bruger en random funktion til at bestemme den endelig værdi. Random funktionen er indbygget i Visual Studio 2010 og fungere således: Random(lavGrænse, højGrænse). Man indtaster den laveste og højeste mulige værdi, og så finder random funktionen et tal derimellem. Random funktionen er baseret på en modificeret version af Donald E. Knuth's subtraktive random tal generator algortime [MSDN, 2011].

Således udregnes tid mellem emner: tidMellemEmner = Random(input rate-25, input rate+25). Der er altså brugt en afvigelse på 25 ms.

Når simuleringen kører ses der ingen bestemte mønstre i hvordan emnerne ligger eller ankommer, som det ses på figuren:



Figur 5.3: Det vurderes umiddelbart, at emnerne ligger tilfældigt uden bestemte mønstre.

Gennemsnitlig ankommer der stadig det antal emner/min og der ankommer den andel 2. sorteringsemner, som er valgt i indstillingerne.

Der er i simuleringen ikke medtaget de kasserede emner, som vil kører igennem anlægget uden at blive samlet op. Da disse ikke skal pakkes, eller muligvis ikke engang

er genkendt af visionsystemet som emner, er det vurderet, at de ikke skal med i simuleringen.

Desuden ligger alle emner i simuleringen ensartet modsat virkeligheden - dvs. de er ikke vendt og drejet på nogen måde. Da udregning af procestiden alligevel ikke medtager det, at skulle dreje End-effectoren, så anses det som unødvendigt at have emnerne til at ligge i forskellige retninger.

Robotcelle

Robotcellen i simuleringen består af en udvælgelse af emne og kasse til næste pick-and-place operation, udregning af procestid for disse bevægelser, som en ruteplanlægger.

For at spare CPU-kraft og undgå fejl, er der en masser kontroller i disse funktioner. Der bliver altid kontrolleret om der er kasser af de rigtige typer, før der kigges efter i emnelisterne efter potentielle emner. Når emne og kasse bliver fundet, som opfylder alle kriterierne fra pakkestrategien, kontrolleres det også, at emnet eller kassen ikke kører ud af arbejdsområdet. Robotten vil således aldrig kører efter et emne eller kasse, den ikke kan nå.

Beregningen for procestiden sker på baggrund af en antagelse om, at robotten bevæger sig konstant accelerende mod emnet/kassen i xy-planet. Der er altså ikke medregnet en opbremsning tid eller bevægelser i z-aksen. På den måde får man udregnet procestider, som er afhængige af afstanden, hvilket var vigtigt for simuleringen. Med den nuværende standard indstilling for accelerationen og dimensionerne bliver procestiderne udregnet til 4-600 ms, hvilket procestiderne også er i virkeligheden. Der kunne dog godt komme en bedre model for procestiden ind i simuleringen - en model, som er baseret på de kinematiske egenskaber for robotten ville være det bedste. Dette bør også være mulig at indarbejde i den nærmeste fremtid, da den nuværende procestidsfunktion foretager sin udregning på baggrund af to punkter og båndets hastighed, ProcesTid(x1,y1,x2,y2,hast).

På trods af at procestiderne ikke er korrekt regnet ud, vil man stadig få nogle brugbare procestider, som netop giver robotten en fordel ved at opsamle emner, der ligger tæt på, end emner, der ligger langt væk. Dette vil altså derfor give fordele til de pakkestrategier, som vælger at pakke emnerne, når de ligger mest fordelagtigt på båndet - den bedste strategi vil altså få fordele, hvilket er vigtigt for vurderingen af en ny pakkestrategi og et af hovedformålene med dette simuleringssystem.

Med procestiderne udregnet, bliver der planlagt en rute for End-effectoren, som fork-

laret i afsnit 4.12 på side 63. Denne proces minder meget om ruteplanlægningen for en virkelig robot, som dog ganske givet ville regne en anden rute ud - den har f.eks. også z-aksen at tage højde for. Ruteplanlægningen i simuleringen er dog også mest vigtig for visualiseringen af End-effectoren, da procestiden jo allerede er udregnet. Robotten kan således ikke blive forsinket af små banaliteter, når først emnet og kassen er valgt, og procestiden er regnet ud. Der er altså i simuleringen ikke taget højde for små forsinkelser, som f.eks. i robottens vakuumsystem. Dette antages dog til ikke at have stor betydning for simuleringens resultater.

Tid

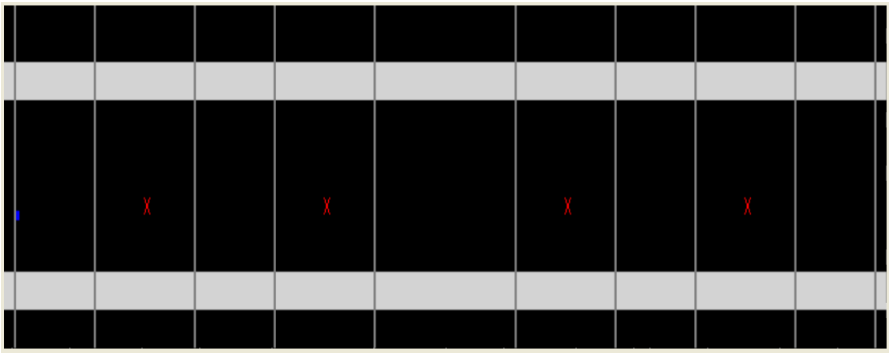
Tiden udregnes, som angivet i afsnit 4.4 på side 49, som Tid = Tid + ΔT per iteration. Der er altså ikke brugt realtid i programmet og det er dermed ikke afhængig af, hvor hurtigt programmet bliver kørt på den pågældende computer. Størrelsen af ΔT bestemmer hvor præcis simuleringen regner - jo større ΔT, jo større fejl vil forekomme. Som standard er ΔT sat til 20 ms, så præcisionen i simuleringen bliver dermed automatisk inden for denne størrelse, hver gang noget er afhængig af tiden, som f.eks. procestiden for en pick-and-place operation. Enkelte funktioner tager højde for den fejl, der kan opstå ved at alting skal gå op med ΔT, f.eks. Input-funktionen. Input-funktionen indsætter nye emner og kasser ifht. input rater. Hvis der skal gå 110 ms til næste emne sættes ind, så går der 120 ms før funktionen udføres(tallet skal gå op med ΔT, 20 ms) - denne fejlmargin på 10 ms bliver regnet ud og tiden, til at næste emne bliver sat på båndet, bliver 10 ms mindre. Der kompenseres altså for fejlmarginet, hvilket har været nødvendigt for at antal emner/min passer.

5.2.2 Test af hele programmet

Som test af programmet som helhed er der foretaget en observering af simuleringen i funktion, hvor der blev holdt øje med at alle de forskellige delkomponenter fungerer sammen.

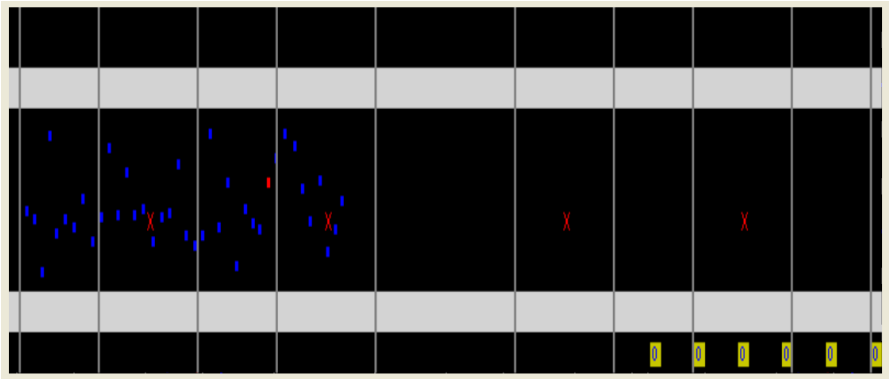
På figur 5.4 på modstående side ses simuleringen, når den lige er startet op. Som det ses er der 4 robotceller med mellemområder mellem sig, hvor det midterste mellemområde er størst, da der her er en controllerboks. X'erne er End-effectorene(EE), som korrekt er startet ca. midt i robotcellernes arbejdsområde.

På figur 5.5 på næste side er der nu gået 6 sekunder og man begynder at kunne obser-



Figur 5.4: Simuleringen er her lige startet, End-effectorne står klar til at samle emner op.

vere emnerne, som kører ind på visionbåndet. De virker pænt fordelt på båndet uden noget egentligt mønster. Blå emner er 1. sorteringsemner, røde er 2. sorteringsemner - der er pt. kun ét enkelt 2. sorteringsemne, indstillingen er sat til 10%, så der burde være flere. Det viser sig dog også at være tilfældigt - en random generator er involveret - og der ankommer mange flere røde emner, senere i observationen. De 10% overholdes.

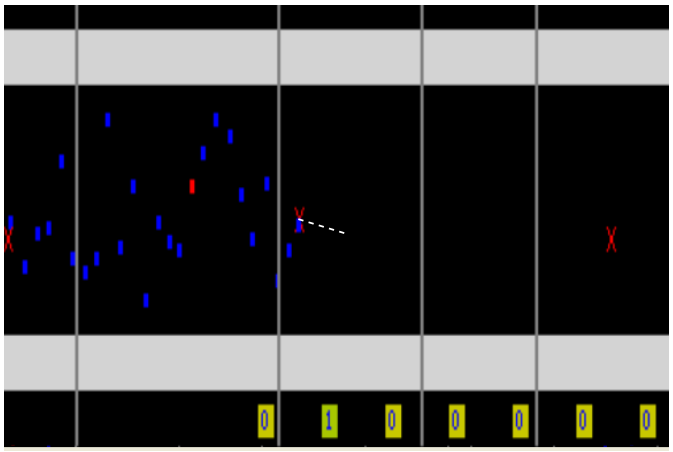


Figur 5.5: Emner bevæger sig lineært hen ad visionbåndet. De er pænt spredt og fordelt, som ønsket.

Det observeres også at emnerne kører korrekt hen ad visionbåndet, Δencoder-konceptet virker altså korrekt.

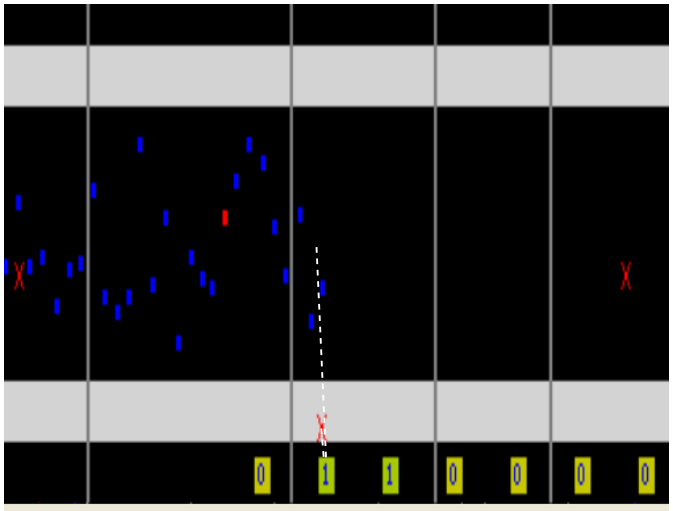
For neden i billedet ses kasserne kommer fra højre på kassebåndet. Disse kører ligeledes korrekt fremad. Foreløblig er ingen EE begyndt at køre efter emner eller kasse, så restriktionerne indtil videre virker fint.

På figur 5.6 på den følgende side ses det, at der i robotcelle 3's arbejdsområde er ankommet emner, som kan pakkes i kasser. Det observeres at EE'eren kører hen til emnet, og da den er lige over emnet, forsvinder denne fra båndet. Dette betyder at programmet korrekt har fjernet emnet fra robotcellernes liste over ledige emner. EE'en fortsætter



Figur 5.6: End-effectoren kører korrekt til emnet og samler den op.

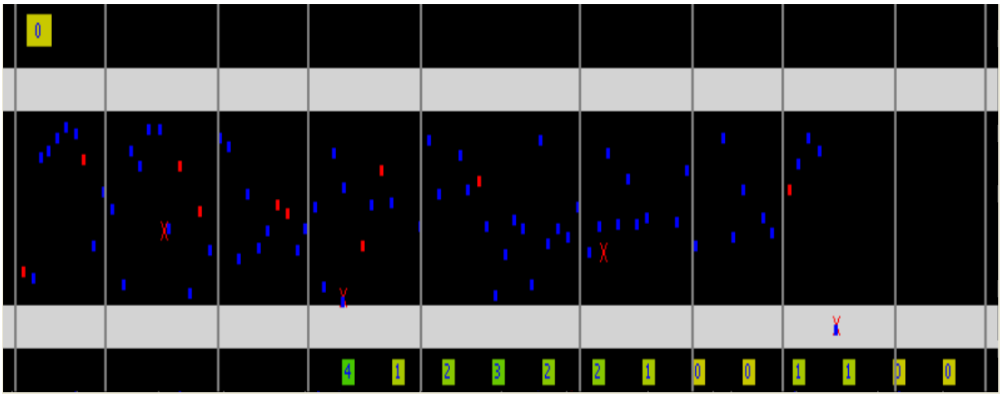
korrekt hen til en ledig kasse og pakker emnet, se figur 5.7, hvor emnet lige er sat i kassen. Under denne kørsel fra EE'en er det observeret at den rammer både emnet og kassen på de helt rigtige tidspunkter - dette viser at både procestid og ruteplanlægningen fungerer korrekt. Det kan yderligere ses, at kassen, som fik emnet, har fået værdien 1 - dette viser, at den nu har et emne i sig, og dermed har en ledig plads, target, mindre.



Figur 5.7: End-effectoren har lige afsat emnet i kassen og er på vej til næste emne. End-effectoren rammer emne og kasse præcist, tegn på at procestid og ruteplanlægning regnes korrekt ud.

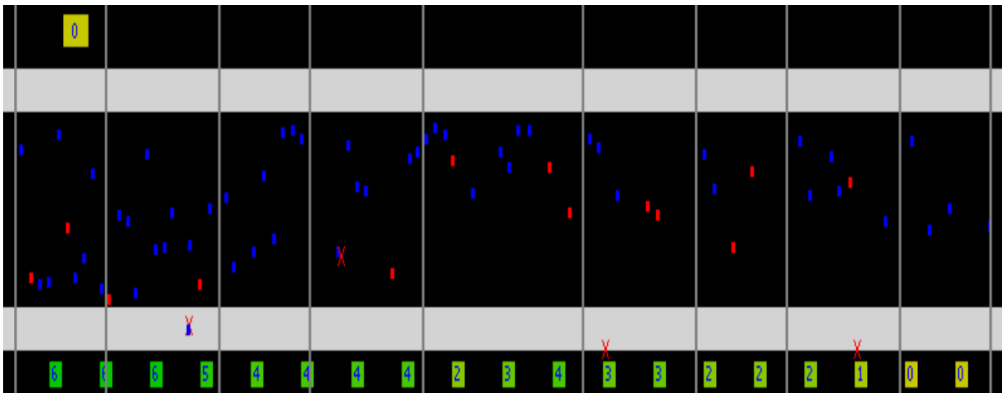
Det kan også ses, at kassen ved siden af har fået tildelt værdien 1. Dette skyldes, at det allerede nu er blevet planlagt, at denne kasse skal modtage næste emne.

På figur 5.8 har simuleringen nu kørt i yderligere nogle sekunder. Der er nu emner og kasser i tre af robotcellerne og EE'erne i disse celler er nu alle igang med at pakke emner. Det observeres at prioriteringsreglerne virker - de celler som skal pakke SPT tager de nærmeste emner, LPT tager de fjerneste emner, FIFO tager emnerne længst til højre og LIFO pakker emnerne længst til venstre. Ved senere observationer er der også forsøgt at bekræfte F-SPT, M-SPT og MF-SPT - disse er dog svære at bekræfte ved observation, da de for det meste kører som almindelig SPT og specielle situationer skal opstå, før de handler anderledes. De er dog delvist bekræftet at virke korrekt - om de giver en fordel ifht. SPT vil blive undersøgt ved test af pakkestrategier i afsnit 6 på side 77.



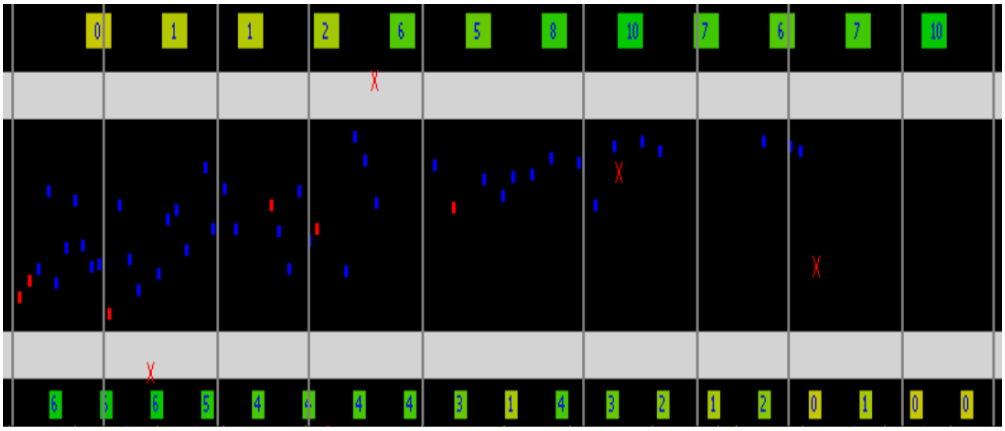
Figur 5.8: Tre af robotcellerne er nu igang med at pakke emner. De pakker korrekt efter deres valgte strategier.

Ved denne observation bekræftes det at alle foranstaltninger og begrænsninger virker mht. at EE'erene ikke prøver at opsamle emner eller kasser, som de ikke kan nå eller som allerede ligger udenfor arbejdsområdet. Det bekræftes desuden at begrænsningen med antal ledige targets virker korrekt, hvilket kan ses på figur 5.9 på den følgende side. Celle 2 og 3 er begrænset til at efterlade 2 targets per kasse - og da der maksimalt kan være 6 emner i en kasse, må der altså maksimalt pakkes 4 emner i en kasse i disse celler. Som det ses, kører alle kasser ud af celle 2 med netop 4 emner i sig - EE'eren i denne celle pakker derefter kun 2. sorteringsemner i kasserne foroven. Det bekræfter samtidig at 2. sorteringsemner virker efter henseende - de bliver kun pakket, når robotterne har tid til overs. Det bekræftes senere i observationen, at robotcelle 1 tager emner øverst i arbejdsområdet, som konsekvens af at vertikal prioritering er valgt til denne celle - den skal netop prioritere emner længst væk fra de små kasser.



Figur 5.9: End-effectorene holder sig inden for arbejdsområderne. Det ses desuden at begrænsning på antal ledige targets virker, da robotcelle 2 efterlader kasser med 4 ledige targets. Desuden virker vertikal prioritering, da robotcelle 1 foretrækker emner længst væk fra de små kasser.

På figur 5.10 er tiden nu oppe på 3 min 47 sek. Dette skyldes at der er blevet 'spolet' frem i tiden - man kan ændre på hastigheden af simuleringen med PilOp- og PilNed-tasterne på tastaturet. Dette bekræftiger, at det er muligt at ændre på hastigheden, hvormed simuleringen bliver afviklet. Det er senere blevet testet, at der ingen forskel forekommer på resulater el. lign. uanset om der blevet kørt med 1x eller 1000x.



Figur 5.10: Simuleringen er her spolet 3 min 47 sek frem i tiden og det bekræftiges at man kan skrue op og ned for simuleringshastigheden uden der opstår fejl.

Denne observation har valideret, at programmet som helhed fungerer, delkomponenterne arbejder sammen og ingen fejl opstår. Der har været løbende observationer som denne test, f.eks. hver gang et nyt element er blevet tilføjet til programmet. Da der pt. ikke kan observeres flere fejl ved programmet, er der nu grundlag for at udføre tests af pakkestrategier.

6 Test af strategier

Dette afsnit indeholder en beskrivelse af en række test, som er blevet udført med simuleringen. Der er blevet testet en langt række forskellige pakkestrategier, og det bliver konkluderet hvilke af disse strategier, der får pakkeanlægget til at klare sig bedst mht. de kriterier, som blev opstillet i afsnit 1.1 på side 3, nemlig fordeling af udnyttelsesgrad, antal pakket emner og kasser, og antal ikke-pakket emner og kasser.

For at afprøve strategierne opstilles der to forskellige scenarier - Testscenarie 1, som indeholder én type emner og kasser og Testscenarie 2, som indeholder to typer. Begge scenarier skal testes af i tre forskellige belastningsgrader kaldet A,B og C, hhv. lav, mellem og høj belastning.

Som beskrevet i afsnit 2.4.5 på side 33 består en pakkestrategi af tre regler:

- Heuristiske prioriteringsregler
- Vertikal prioritering
- Antal frie targets

I den første testfase vil pakkestrategierne være simple sat op, da der hverken vil være vertikal prioritering eller valgt et minimum antal frie targets. De heuristiske prioriteringsregler vælges for alle robotcellerne og vil alle blive prøvet af. Dvs. at der vil være 7 pakkestrategier, som skal prøves af, med hhv. FIFO, LIFO, SPT, LPT, M-SPT, F-SPT og MF-SPT.

I den anden testfase testes der pakkestrategier, som består af forskellige kombinationer af alle tre regler. I alt bliver der testet 5 forskellige strategier i denne testfase.

Slutteligt vil en konklusion følge op på, hvilke pakkestrategier, der passer bedst til de forskellige scenarier og opfylder de opstillede kriterier bedst.

Alle scenarier vil indeholde 4 robotceller og strategierne testes af i en 8 timer simulering. De generelle indstillinger, som er ens for alle forsøg, er opstillet i tabel 6.1 på den følgende side.

Indstilling	
Visionbåndets hastighed	50 %
Kassebånd 1's hastighed	40 %
Kassebånd 2's hastighed	5 %
Robotternes acceleration	20 %
Bredden af arbejdsområdet	90 cm
Længden af arbejdsområdet	90 cm
Bredden af kontrolboksen	55 cm
Afstand mellem arbejdsområder	72 cm

Tabel 6.1: Her ses de generelle test indstillinger som er ens i alle forsøg. Bemærk at kassebånd 2's hastighed kun anvendes i Test scenarie 2.

6.1 Første testfase

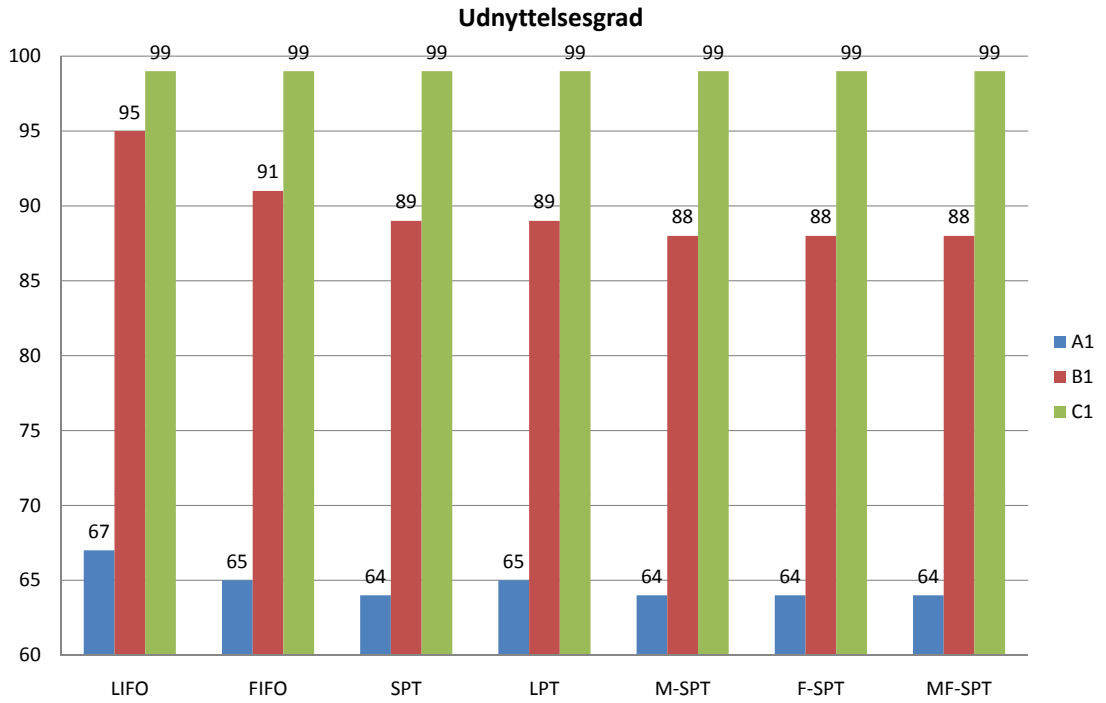
6.1.1 Test scenarie 1

I Test scenarie 1 opstilles simuleringsforsøgene i tre niveauer, som angives A1, B1 og C1. Dette er med hhv. lav, middel og høj belastning i form af indkomne emner og kasser, som kan ses i tabel 6.2.

	Emne input rate[1/min]	Kasse input rate[1/min]	Antal targets
A1	300	50	6
B1	420	70	6
C2	540	90	6

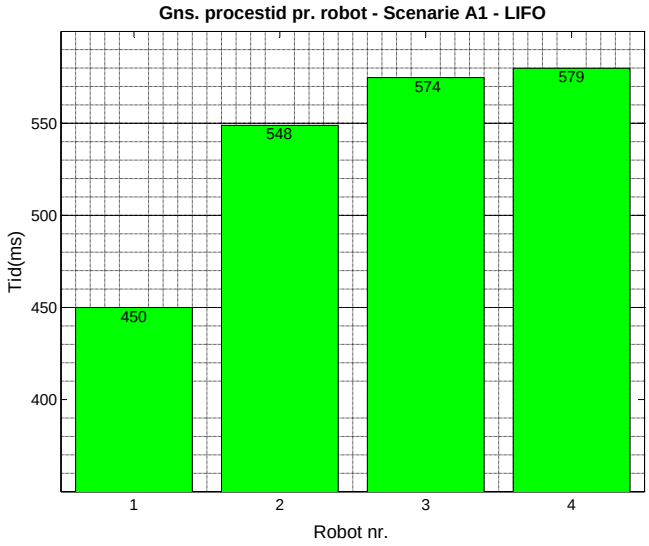
Tabel 6.2: Her ses de tre test niveauer A1,B1 og C1, som er hhv. lav, middel og høj belastning.

Med 7 forskellige strategier og 3 belastningsniveauer bliver der for Testscenarie 1 altså 21 forsøg, som i det efterfølgende vil blive analyseret. Det første som analyseres er den samlet udnyttelsesgrad for hele pakkeanlægget. Dette er gennemsnittet af udnyttelsesgraden for de fire robotter og kan ses på figur 6.1 på modstående side.

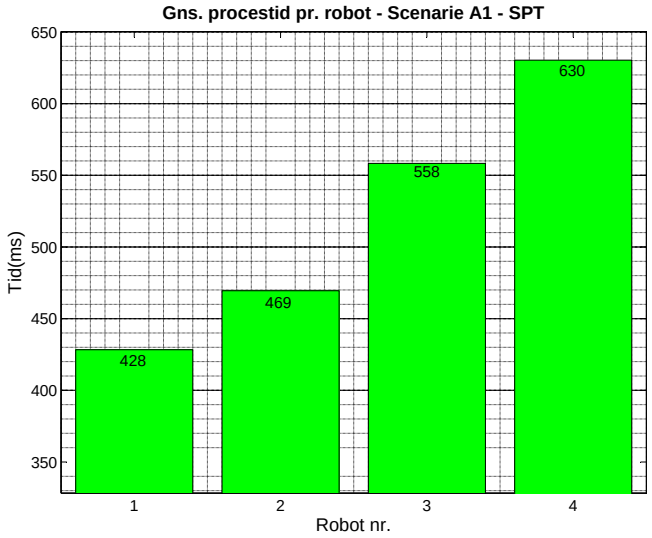


Figur 6.1: Her ses udnyttelsesgraden for anlægget med de 7 strategier for de tre belastningsniveauer A1, B1 og C1.

Det kan ses at i scenarie A1 ligger udnyttelsesgraden ved de forskellige prioriteringsregler mellem 64-67%, hvor LIFO er et par procentpoint bedre end de andre. Da der er meget lav belastning i dette scenarie, betyder det at robotterne ofte kommer til at stå stille. Derfor vil en højere gennemsnitlig procestid betyde, at udnyttelsesgraden vil være højere. Dette er netop hvorfor LIFO har højere udnyttelsesgrad - se evt. afsnit 2.4.4 på side 29 om, hvorfor denne har længere procestider. De gennemsnitlige procestider for LIFO kan ses på figur 6.2 på den følgende side og kan sammenlignes med procestiderne for SPT på figur 6.3 på næste side. I scenarie B1 er udnyttelsesgraden på 88-95%, hvor LIFO igen skiller sig ud med klart højeste udnyttelses, igen pga. sine længere procestider. I scenarie C1 er belastningen på anlægget så højt, at alle robotter konstant arbejder, og udnyttelsesgraderne er derfor alle på 99%.

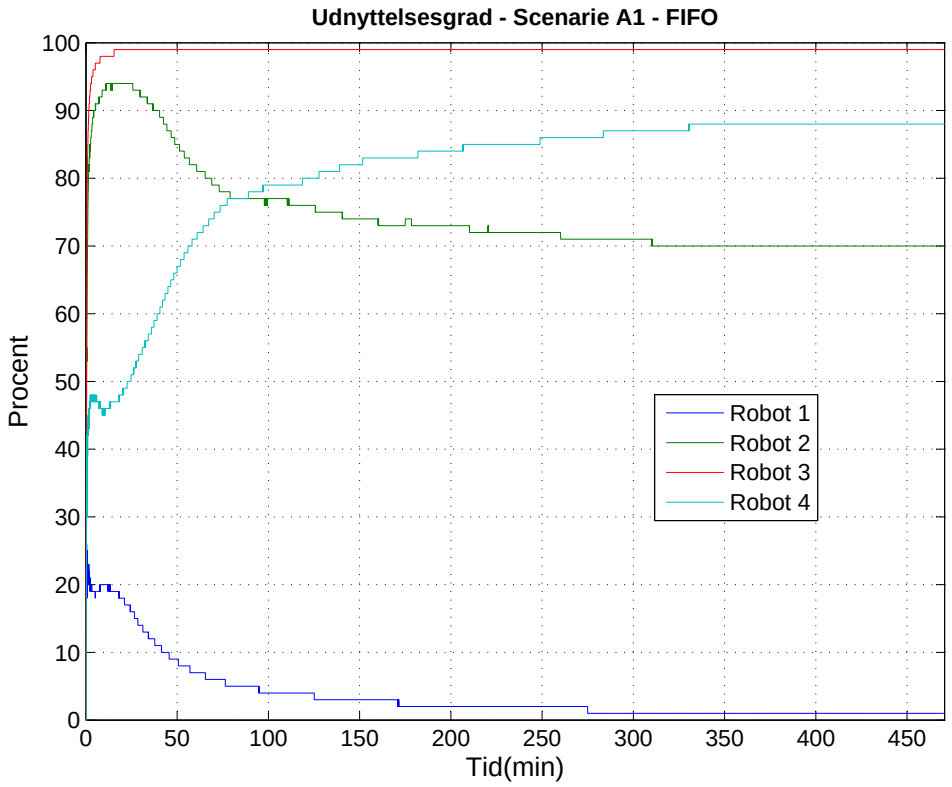


Figur 6.2: Gns. procestider for LIFO i scenarie A1.



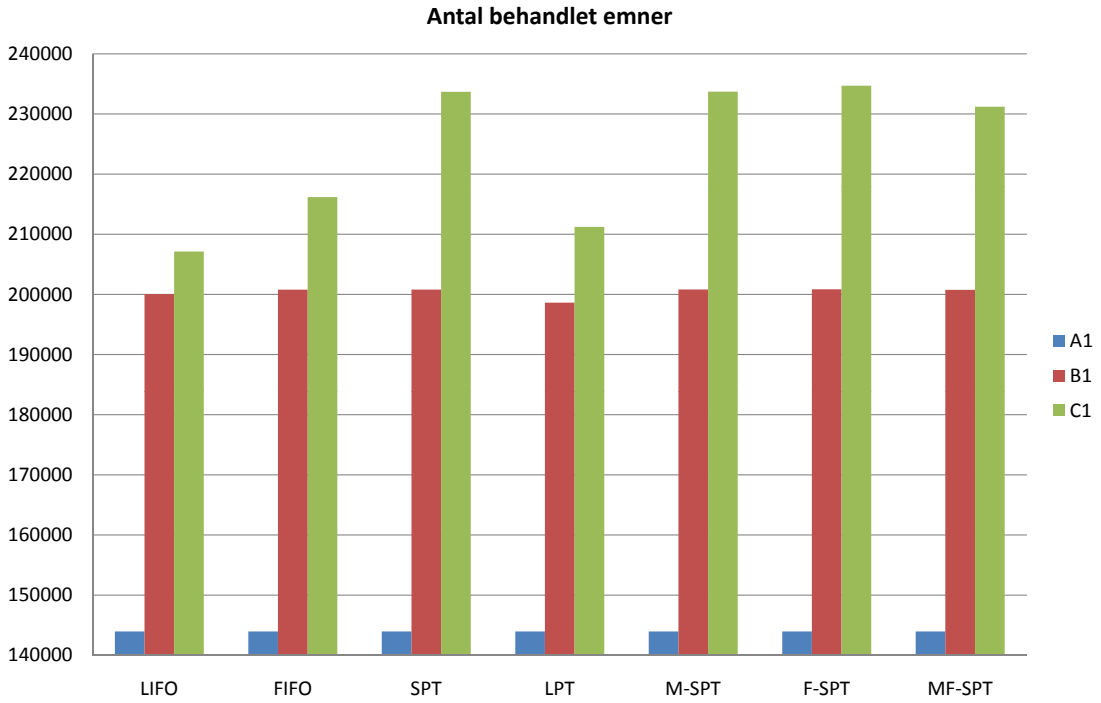
Figur 6.3: Gns. procestider for SPT i scenarie A1. Bemærk at den gns. procestid næsten stiger linenært for robotterne, hvilket skyldes at emner med kortest procestid behandles først.

Den lave udnyttelsesgrad i A1, som ses på figur 6.1 på forrige side, skyldes som nævnt for lav belastning på pakkeanlægget. Dette medfører at udnyttelsesgraden for enkelte robotter falder helt i bund i løbet af simuleringsperioden, hvilket kan ses på figur 6.4 på næste side. Udnyttelsesgraden for Robot 1 bliver næsten 0%, hvilket skyldes at kasserne for det meste er fyldte, når de ankommer til denne robotcelle.



Figur 6.4: Her ses udnyttelsesgraden for de fire robotter ved scenarie A1 med FIFO. Robot 1 får den gennemsnitlige udnyttelsesgrad til at falde betragtelig, hvilket skyldes at kasserne ofte er fyldte, når de ankommer til robotcelle 1.

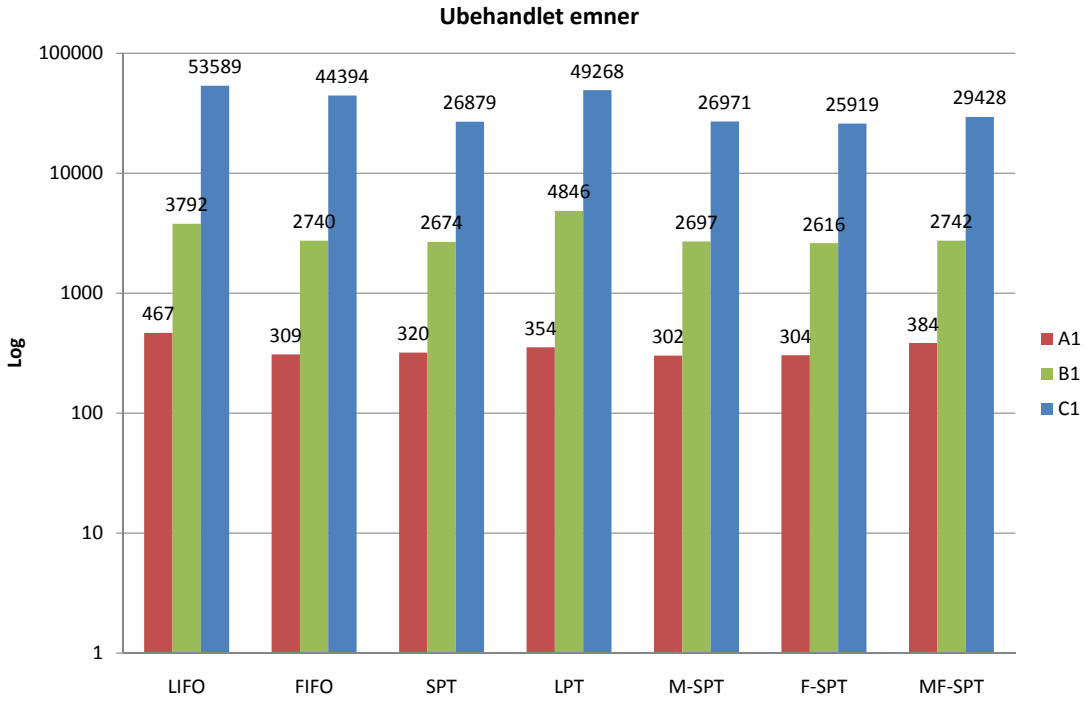
En høj udnyttelsesgrad for en pakkestrategi er ikke nødvendigvis lig med at der pakkes mange emner. Grunden til dette er, at nogle prioteringsregler prioritere emner med en høj procestid, og derved er robotterne beskæftiget i længere perioder, uden at pakke flere emner. Alle prioriteringsreglerne har pakket i 8 timer, og derfor er det mere interessant at se på hvilken pakkestrategi, der har medført flest pakkede emner.



Figur 6.5: Her ses antal behandlet emner over 8 timer ved de forskellige scenarier A1,B1 og C1.

I løbet af de 8 timers produktion vil der løbe ca. 144.000, 202.000 og 260.000 emner igennem anlægget ved henholdsvis scenarie A1, B1 og C1. På figur 6.5 ses det at pakkestrategierne pakker lige mange emner ved lav og middel belastning, hvilket skyldes at stort set alle emnerne bliver pakket. Det er først i scenarie C1, hvor belastningen stiger til 540 emner/min, at der ses en tydelig forskel mht. antal pakkede emner. Her er det klart prioriteringsreglerne SPT, M-SPT, F-SPT og MF-SPT der pakker flest. Dette skyldes at SPT reglerne altid vil pakke de emner, som har den mindste procestid først, og derved lade de emner med store procestider kører ud af anlægget. Det ses også at LIFO, som havde den højeste gennemsnitlige udnyttelsesgrad, pakker færrest emner og derved er mindst effektiv.

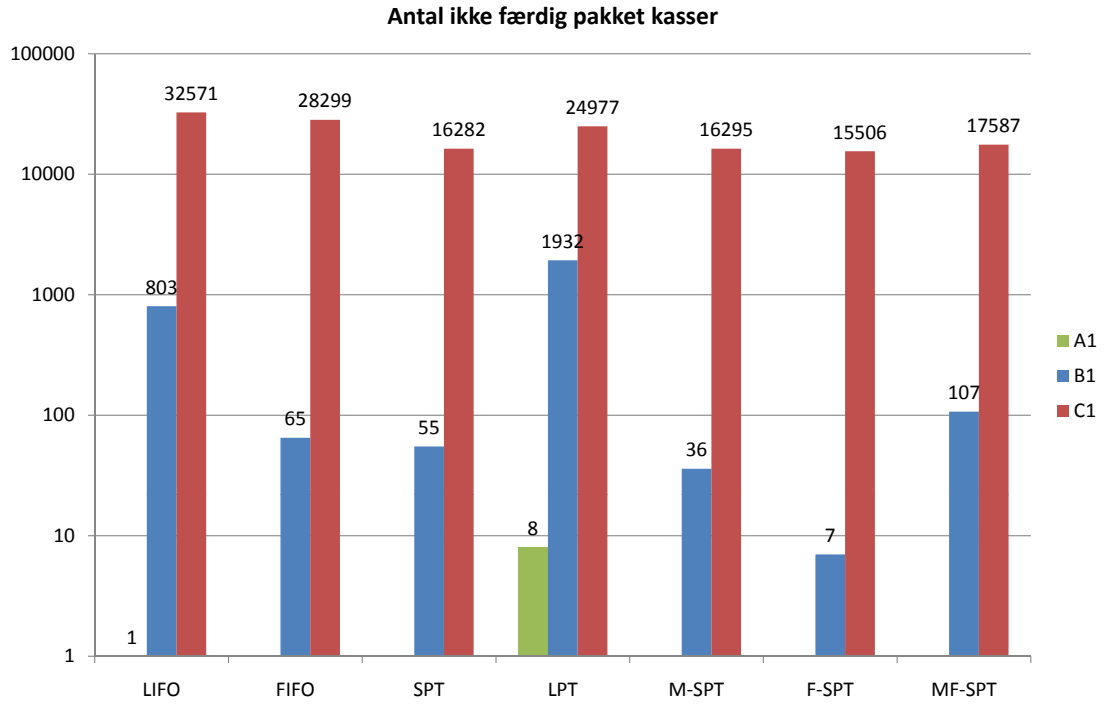
Et andet kriterie er antallet af ubehandlede emner, som skal minimeres. Derfor er det interessant, at se hvordan pakkestrategierne performer i forhold til dette kriterie, hvilket kan ses på figur 6.6.



Figur 6.6: Her ses antal ubehandlet emner for hver scenarie A1, B1 og C1. Det ses at pakkestrategierne indenholdende SPT princippet og FIFO lader færrest emner kører ud af anlægget.

Det kan ses at scenarie C1 er alt for høj belastning for pakkeanlægget, hvilket medfører at det der langt fra kan pakkes alle emner. Dette giver et meget højt antal ubehandlet emner i forhold til de andre scenarier, men samtidigt er det interessant at se hvilke prioriteringsregler der performer bedst under disse høje belastninger. Det er klart, at de prioriteringsregler, som pakker flest emner, også burde have det mindste antal ubehandlet emner. Dette kan også ses på figur 6.6 hvor SPT-gruppen har minimum antal ubehandlet emner i alle tre scenarier. En anden interessant observering er, at se FIFO performer på samme niveau, som SPT, F-SPT og M-SPT i scenarie A1 og B1, hvilket dermed kan konkluderes, at FIFO er en god prioriteringsregel op til en hvis belastning mht. at minimere antal ubehandlet emner. Dette er forventet, da FIFO netop er god i den sidste robotcelle, 4, til at tage emnerne, inden de ryger ud af anlægget.

En af kriterierne, som ønskes minimeret, er antallet af ikke færdig pakket kasser, som forlader pakkeanlægget. I otte timers produktion vil der komme 24.000 kasser, 33.600 kasser og 43.200 kasser gennem pakkeanlægget for hhv. scenarie A1, B1 og C1. Derfor er det interssant at se, hvor mange af disse, som ikke når at blive fyldt op inden de forlader pakkeanlægget, hvilket kan ses på figur 6.7.



Figur 6.7: Her ses antal ikke færdig pakket kasser for scenarie A1, B1, og C1. I scenarie A1 når alle pakkestrategier at pakke alle kasser på nær LIFO og LPT, hvorved disse kun er plottet.

På figur 6.7 kan ses at i scenarie A1 bliver alle kasser stort set pakket. Ved B1 falder LIFO og LPT ud som de store syndere. Ved LIFO skyldes det *ikke*, at den ikke kan følge med - den har ca. samme procestider som FIFO. Det skyldes derimod, at Robot 1 prioritere de kasser, som kommer sidst ind i arbejdsområdet. Dvs. hvis der kommer 2-3 kasser ind, som alle mangler emner, prioriteres den bagerste kasse, og den første kasse vil køre igennem uden at blive fyldt op. Ved LPT følger anlægget også pænt med og pakker mange emner, men problemet opstår igen ved robotcelle 1. Den står ofte og laver ingenting, har lav udnyttelsesgrad, men har de længste procestider af alle robotceller, da den har flest emner. Så når der endelig dukker et par kasser op, som skal fyldes, kan den kun nå at pakke én af kasserne. I scenarie B1 er F-SPT klart bedst, da der kun er 7 kasser, som ikke er fyldte - den havde også færrest ubehandlet emner.

Herefter kommer de andre i SPT-gruppen, som alle på nær MF-SPT ligger omkring de 30-60 kasser. Grunden til denne SPT-gruppe klarer sig bedst, er at Robot 1 vil have flest emner at vælge imellem, og derfor også har de laveste procestider. Dvs. at den nemt at nå at pakke de sidste kasser, selvom der måske skulle komme en lille bølge af kasser, som mangler emner. Det er igen værd at bemærke, at FIFO ligger meget tæt på SPT-gruppen og er bedre end MF-SPT. Dette skyldes, at Robotcelle 1 vil prioritere de første kasser, altså dem som er tættest på at køre ud af anlægget. En konklusion kunne da være, at det ville være en god idé at køre FIFO på kasserne kombineret med SPT for emnerne i Robotcelle 1.

6.1.2 Test scenarie 2

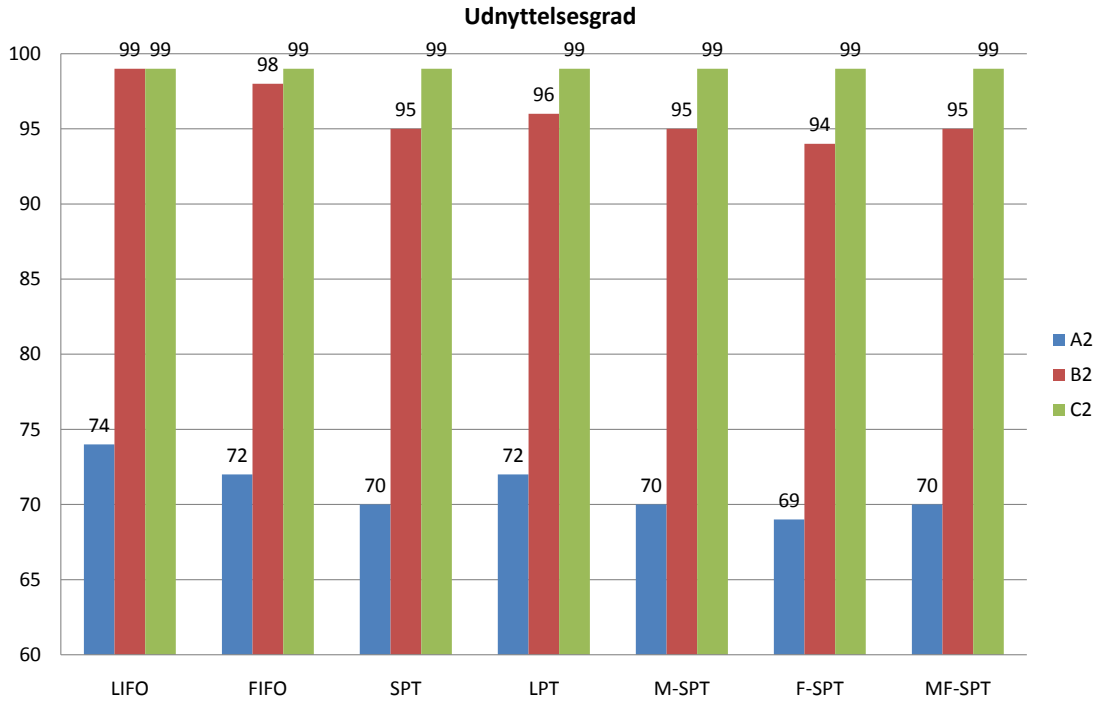
I Test scenarie 2 opstilles simuleringsforsøgene igen i tre belastnings niveauer, angivet som A2, B2 og C2. I Test scenarie 2 indføres der 2. sorteringsemner(type 2 emner). Dvs. det øverste transportbånd med store kasser kommer nu i brug. Dette transportbånd, som tidligere fortalt, har samme flowretning som emne transportbåndet. Hastigheden på det store kassebånd er 5%, hvor det lille kassebånd er 40%, se evt. tabel 6.1 på side 78. De resterende indstillinger for de tre test niveauer ses i tabel 6.3, hvor der både er input rate for type 1 og 2 for emner og kasser. Type 2 emnerne udgør 10 % af emnerne.

Alle prioriteringsreglerne prioriterer type 1 emner over type 2 emner - da type 2 er tænkt som 2. sorteringsemner. Dette vil sandsynligvis medføre, at der bliver en stor del ubehandlet type 2 emner i nogle af scenarierne, da kapaciteten ved visse belastninger allerede var fuldt udnyttet i Test scenarie 1.

	Emne input rate [1/min]	**Kasse type 1 input rate[1/min]**	**Antal targets**	**Kasse type 2 input rate[1/min]**	**Antal targets**
A2	333	50	6	3	11
B2	466	70	6	4	11
C2	600	90	6	6	10

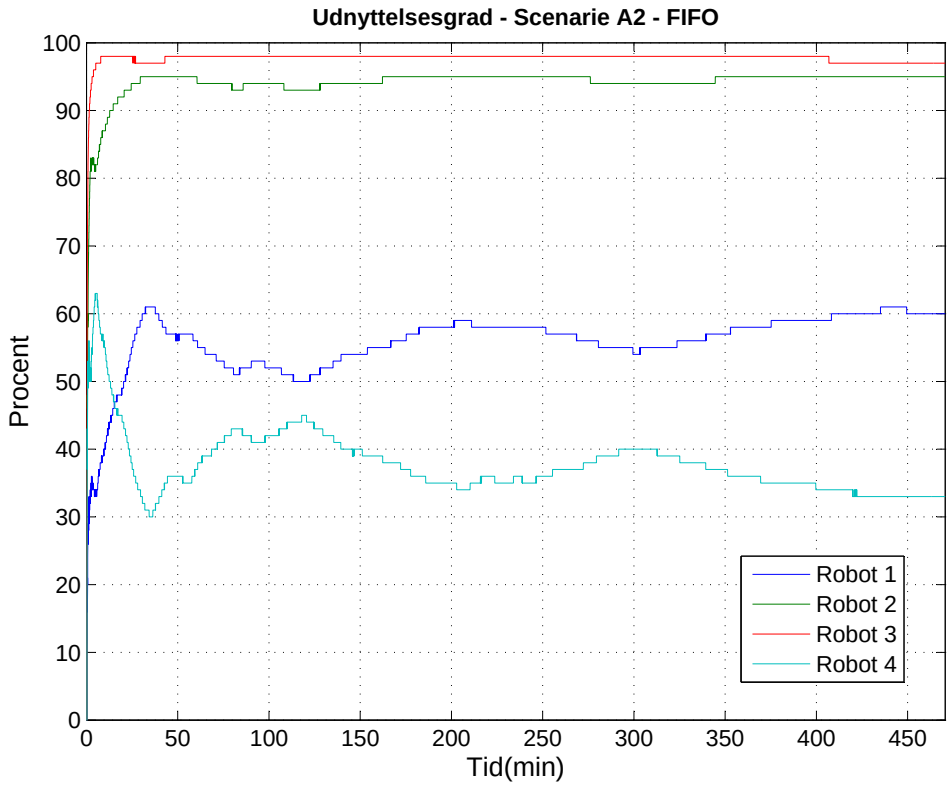
Tabel 6.3: Her ses de tre test niveauer A2, B2 og C2, som er hhv. lav, middel og høj belastning. Bemærk at 2. sorteringsemner udgør 10 % af emnerne, som dermed er lagt til input raten.

Igen er der 7 pakkestrategier og tre belastningsniveauer, så i alt 21 forsøg blev udført med Test scenarie 2. Disse vil i det efterfølgende blive analyseret. Det første som analyseres er den samlet udnyttelsesgrad for hele pakkeanlægget, se figur 6.8 på næste side.



Figur 6.8: Her ses udnyttelsesgraden for scenarier A2, B2 og C2. I scenarie A2 og B2 er disse højere end scenarie A1 og B1, hvilket skyldes at der også pakkes 2. sorteringsemner.

Det er interessant at se udnyttelsesgraden for A2 og B2 er højere end i scenarie 1, hvilket skyldes at robotterne kan pakke 2. sorterings emner, når der ingen 1. sorterings emner er. I Test scenarie 1 blev det observeret på figur 6.4 på side 81, at Robot 1's udnyttelsesgrad efter noget tid falder til næsten 0 %. Dette skyldes, at Robot 2, 3 og 4 når at fylde alle kasserne, og derved har Robot 1 ingen opgaver. Nu har Robot 1 2. sorteringsemner at pakke. Denne effekt kan ses på figur 6.9 på næste side.



Figur 6.9: Her ses udnyttelsesgraden for scenarie A2 med FIFO - bemærk hvordan Robot 1 og 4 påvirker hinanden direkte.

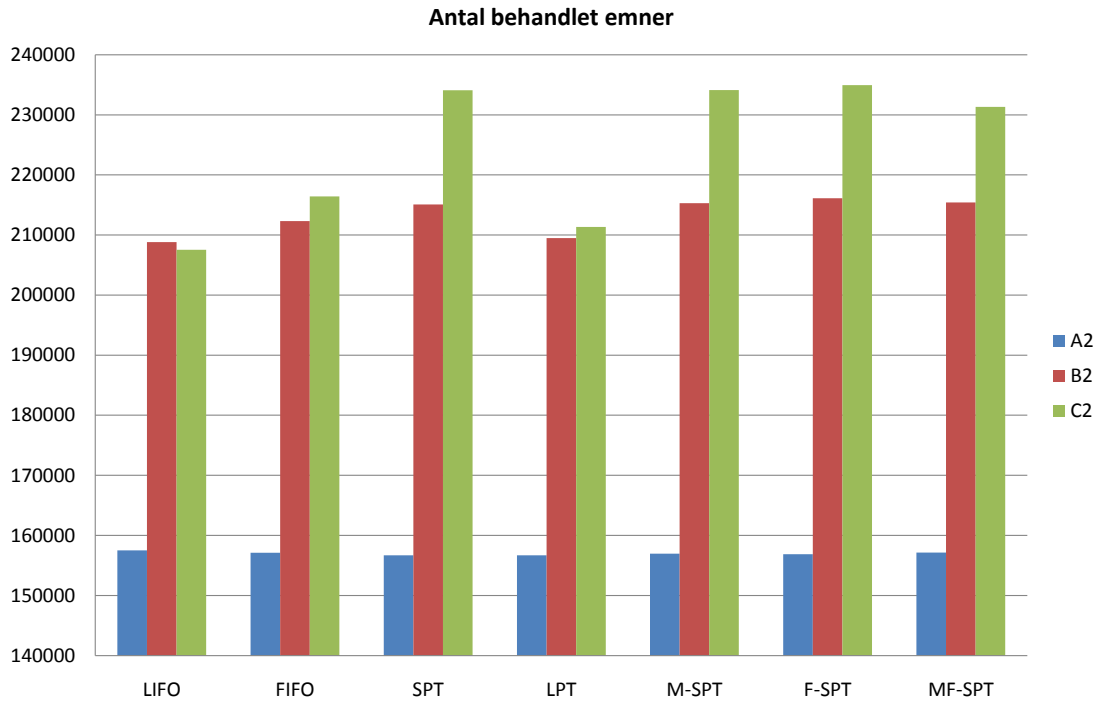
Det observeres at det ikke er de samme robotter, som pakker flest emner i forhold til Test scenarie 1. Her var rækkefølgen for højst udnyttelsesgrad hhv. Robot 3, 4, 2 og 1, som kan ses på figur 6.4 på side 81. Ved at indføre 2. sorteringsemnerne er rækkefølgen nu 3, 2, 1 og 4, hvilket er interessant da Robot 4 nu er den, som har lavest udnyttelsesgrad. Det mest interessante er dog, hvordan Robot 1 og 4 skiftevis falder og stiger modsat hinanden. Dette skyldes, at disse to celler faktisk "kæmper" om at pakke emner. I starten vil Robot 2, 3 og 4 pakke alle type 1 kasser, så Robot 1 kan kun pakke type 2 emner. Men det hænder, at der kommer en "bølge" af type 2 emner, som Robot 1 får pakket. Dette skaber et lille "hul", hvor der ingen type 1 emner er tilbage til Robot 4 at pakke. Dette gør, at der kommer flere type 1 kasser op til Robot 1, som den så kan pakke. Jo flere gange disse bølger kommer, jo flere gange flyttes belastningen over til Robot 1, som nu i samarbejde med Robot 2 og 3, får pakket alle type 1 emner. Dette efterlader kun type 2 emner til Robot 4 at pakke, så nu falder dennes udnyttelsesgrad.

Det viser sig, at alle pakkestrategier ved scenarie A2 har den samme "kamp" om at

pakke emner mellem Robot 1 og 4 - hvor hurtigt belastingen skifter hænder kommer an på pakkestrategien, da det er afhængigt af, hvor effektivt Robot 2 og 3 kan pakke - jo hurtigere de pakker, jo mindre effekt vil en "bølge" have. Scenarie A1 har ikke denne kamp - dette skyldes at der aldrig kommer bølger af type 2 emner, som kan bryde rytmen, og derfor stabiliserer belastingen på robotterne sig hurtigt og fastholdes.

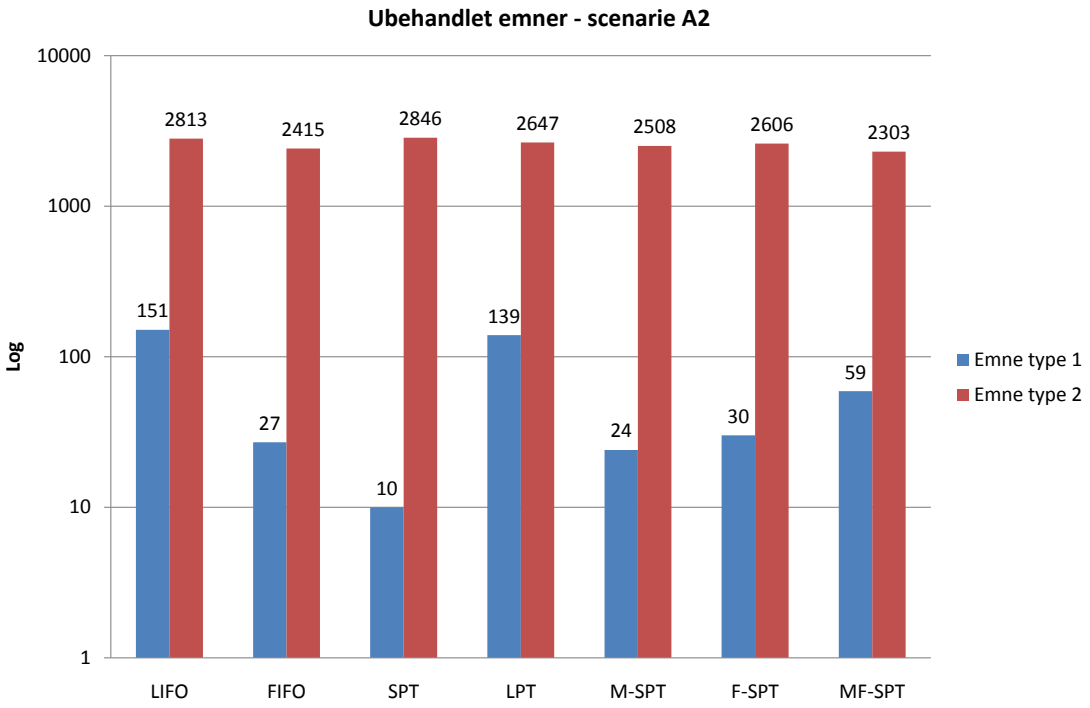
Denne "kamp" er meget interessant, og vil helt sikkert kunne bruges mange timer på at studere.

Kigges der på antallet af behandlet emner, er det igen SPT-gruppen og FIFO som pakker flest, ligesom i Test scenarie 1. Dette giver også god mening, da robotterne prioriterer type 1 emner, der ankommer i samme antal, som i Test scenarie 1. Dermed pakkes type 1 emnerne først og efterfølgende har robotterne tid til at pakke 2. sorteringsemnerne, hvilket øger antallet af pakket emner, som ses på figur 6.10.



Figur 6.10: Her ses det samlet antal behandlet emner i scenarie 1 med belastning A2, B2 og C2.

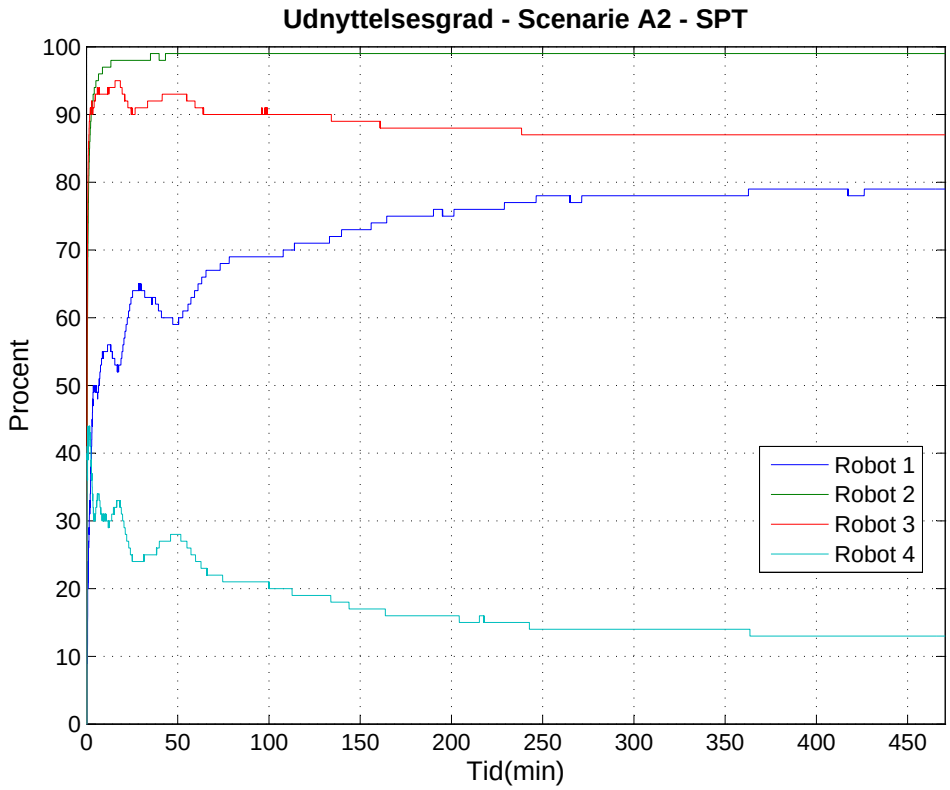
På grund af at tendensen mht. antal behandlet emner er ens i forhold til Test scenarie 1, er det mere interessant at se nærmere på antallet af ubehandlet emner, hvor der i dette Test scenarie 2 er to typer. Det vælges kun at fremvise antal ubehandlet emner fra A2, hvilket skyldes at i scenarie B2 og C2 bliver belastningen for høj, som medfører et højt antal ubehandlet emner af type 1 og alle type 2 emner, da disse stort set ikke behandles.



Figur 6.11: Her ses antal ubehandlet emner fra scenarie A2, som der hhv. er type 1 og 2 emner.

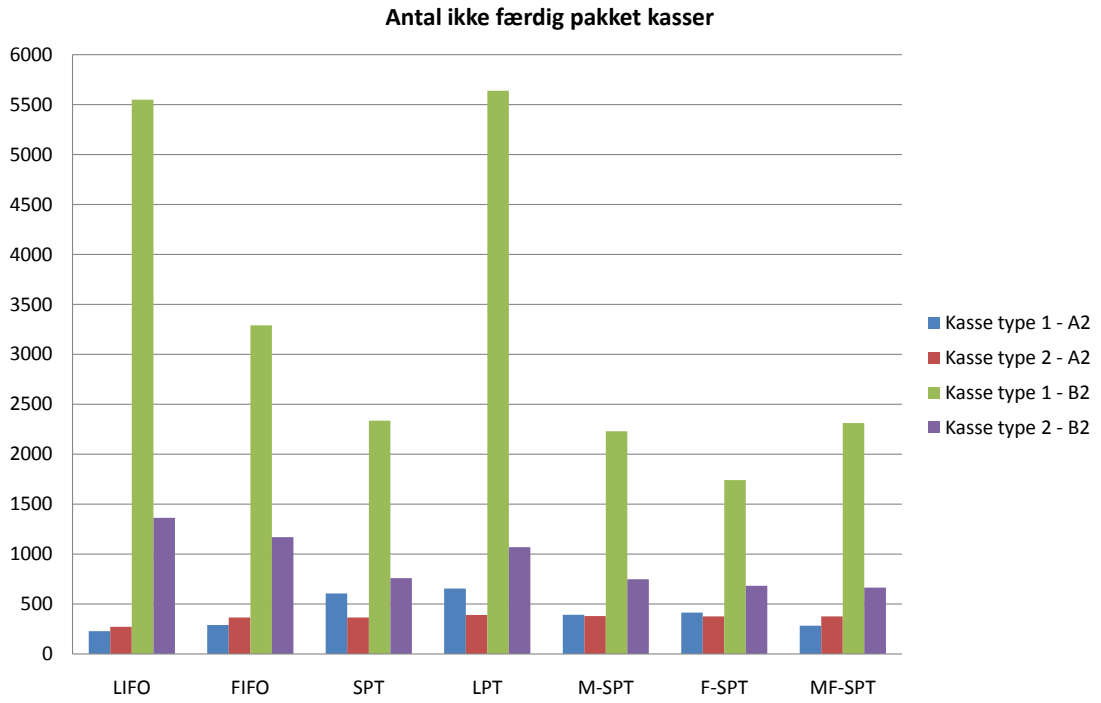
Det er interessant at lægge mærke til at SPT kun har 10 ubehandlet type 1 emner, hvilket er meget lavt i forhold til i Test scenarie 1, hvor SPT havde 320. De andre strategier er ligeledes faldet fra 3-400 ubehandlet emner til nu 10-150 emner. Dette fald skyldes, at 2. sorteringsemnerne har den effekt, at type 1 emnerne kommer en anelse mere spredt ud og danner generelt færre "klumper" - disse grupper af tætpakket emner i Test scenarie 1 kunne robotterne ikke altid nå at pakke, og derfor slap nogen igennem. Når type 1 emnerne nu er mere spredt, kan robotterne nemmere nå at samle dem op.

Dette kan også ses på figur 6.12 på den følgende side, som viser udnyttelsesgraden for scenarie A2 med SPT. Det er tydeligt, at Robot 4 hurtigt falder til en meget lav udnyttelsesgrad og dermed har denne altid kapacitet til, at pakke de sidste type 1 emner og komme dem i de små kasser, som altid har frie targets, da disse ankommer først til robotcelle 4. Det skal dog også observeres, at SPT har det største antal ubehandlet type 2 emner, hvor FIFO, M-SPT, F-SPT og MF-SPT næsten har det samme antal ubehandlet type 1 emner, men et mindre antal af ubehandlet type 2 emner. Dette er interessant da disse prioriteringsregler dermed synes, at være fordelagtige at anvende i den sidste robotcelle, da de netop kan behandle emner, som er tæt på den øvre grænse i arbejdsområdet.



Figur 6.12: Her ses udnyttelsesgraden for scenarie A2 med SPT. Det er tydeligt at se Robot 4 har kapacitet til at pakke de resterende type 1 emner.

Det er også interessant at se på antallet af ikke færdig pakket kasser, da dette helst skal minimeres. I Test scenarie 1 var SPT-gruppen klart bedst, hvor F-SPT var den bedste af disse.



Figur 6.13: Her ses antal ikke færdig pakket kasser, hvor der både er kasse type 1 og 2. Scenarie C2 undlades, da dette giver et rigtigt stort antal ikke færdig pakket kasser pga. den høje belastning på anlægget.

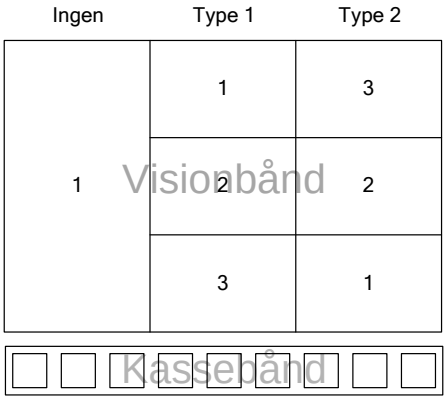
Det ses på figur 6.13 at det igen er SPT-gruppen, som er bedst. Det skal dog også observeres, at FIFO ved lav belastning igen performer på samme niveau, som SPT-gruppen.

6.2 Anden testfase

I den anden testfase ønskes det at prøve nogle kombinationer af de tre pakkeregler af. I første omgang bliver fire kombinationer testet, en speciel femte kombination kommer til sidst i afsnittet.

De tre regler var som tidligere nævnt **heuristiske prioriteringsregler**, **vertikal prioritering** og **minimum antal targets**.

Vertikal prioritering har tre typer, se figur 6.14 - den handler altså om at prioritere emnerne efter, om de ligger langt fra eller tæt på type 1 kasserne.



Figur 6.14: Med det lille kassebånd forneden og visionbåndet foroven, er der her illusteret, hvordan emnerne på visionbåndet kan prioriteres i den vertikale retning. Der er to typer: Type 1 - Her prioriteres emnerne længst væk fra kassebåndet. Type 2 - Her prioriteres emnerne tættest på kassebåndet.

Minimum antal targets, er det antal frie pladser, som minimum skal være tilbage i kasserne, når de forlader et arbejdsområde.

Der opstilles fire pakkestrategier, P1, P2, P3 og P4, som er opstillet i tabel 6.4 på næste side. De er ens på alle punkter på nær, at de bruger forskellige heuristiske prioriteringsregler i robotcelle 4. Her bruger de hhv. FIFO, M-SPT, F-SPT og MF-SPT. Ellers gælder SPT i celle 1, 2 og 3 i alle strategierne.

Robotcelle 2 og 3 har vertikal prioritering type 1, dvs. at type 1 emnerne, som ligger længst væk fra type 1 kasserne, prioriteres højest. Dette skal gerne medføre, at robotcelle 2 og 3 tager de emner, som ligger længst væk, og derved efterlader robotcelle 4 nogle hurtige emner. Dette skal sikre, at den kan nå at tage de sidste emner, inden de forlader anlægget.

Yderligere er der sat et minimum antal ledige targets på type 1 kasserne. Dette skal

fordele belastningen til hele anlægget og sikre, at Robot 2 og 3 ikke pakker alle emnerne. Det vil samtidig give disse to robotter frirum til at tage sig af type 2 emnerne. Ydermere skulle det gerne sikre, at kasserne bliver fyldt mere jævnt op. Dvs. at der ikke er kasser, som ankommer til Robot 1 og er fyldt med f.eks. 5 emner i én kasse og 1 emne i en anden kasse - denne situation gør det nemlig svært for Robot 1 at nå at få pakket den sidste kasse, inden den forlader anlægget.

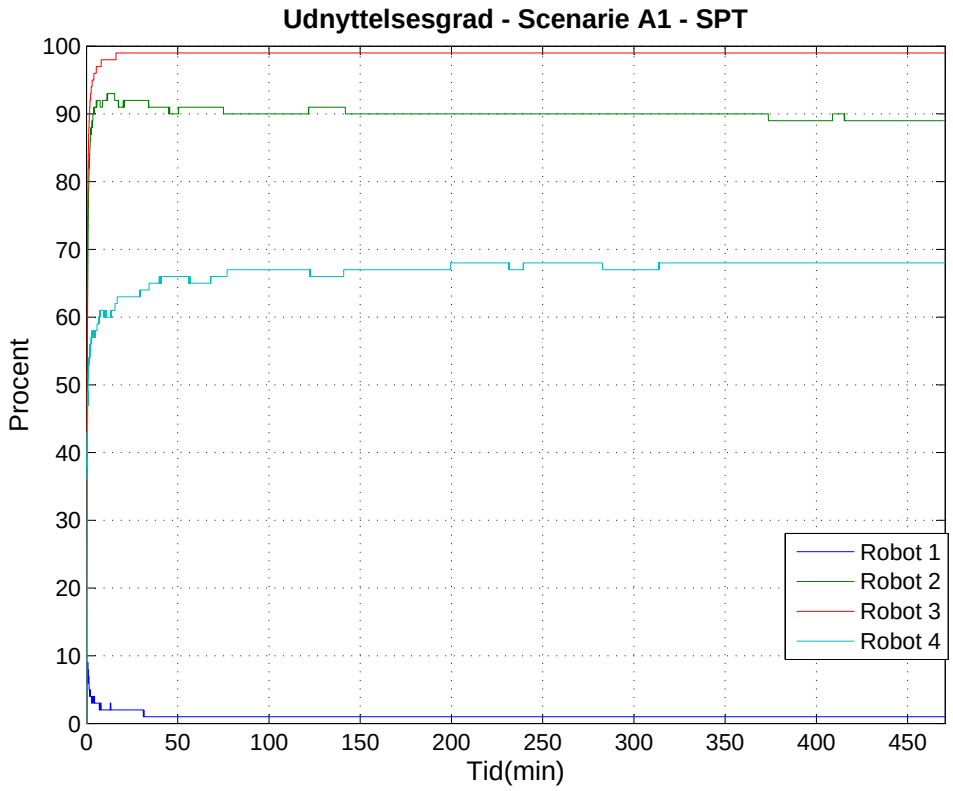
	Robotcelle 1	**Robotcelle 2**	**Robotcelle 3**	**Robotcelle 4**
Prio. regel	SPT	SPT	SPT	FIFO, M-SPT, F-SPT, MF-SPT
Vert. Prio.	Ingen	Type 1	Type 1	Ingen
Min. targets	0	2	3	4

Tabel 6.4: Her ses de valgte **heuristiske prioriteringsregler**, **vertikal prioritering** og **minimum antal targets** for de fire pakkestrategier.

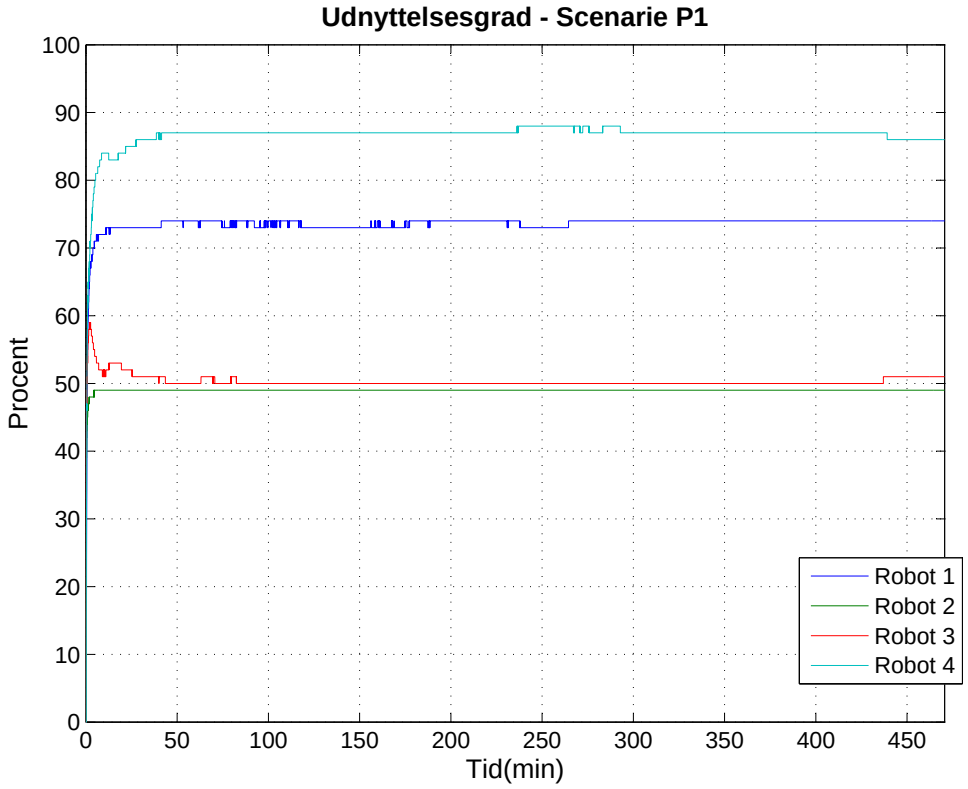
Til at teste pakkestrategierne anvendes igen et Test scenarie 1 og 2, som har hhv. ét og to typer emner. Indstillinger som anvendes er de samme som i Testfase 1, og kan ses i tabel 6.2 på side 78 for A1, B1 og C1 og i tabel 6.3 på side 86 for A2, B2 og C2. Dermed udføres 24 forsøg.

6.2.1 Test scenarie 1

Resultaterne fra de første 12 forsøg med A1, B1 og C1 er ikke meget forskellige fra forsøgene med SPT i Testfase 1, hvilket helt naturligt skyldes at P1, P2, P3 og P4 jo netop bruger SPT i alle celler undtagen Robotcelle 4. De vil derfor pakke ca. lige mange emner. Der, hvor der gerne skulle være forskel, er i fordelingen af arbejdet og ikke-pakket emner og kasser. Fordelingen af arbejdet bør være bedre pga. vertikal prioritering og minimum antal targets. Dette kan ses på de næste to figurer, hvor figur 6.15 er udnyttelsesgraden for SPT fra Testfase 1 ved A1 og figur 6.16 på næste side er udnyttelsesgraden for pakkestrategi P1. Udnyttelsesgraderne er gennemsnitlig på hhv. 64% og 65%, men P1 har fordelt arbejdet meget bedre.

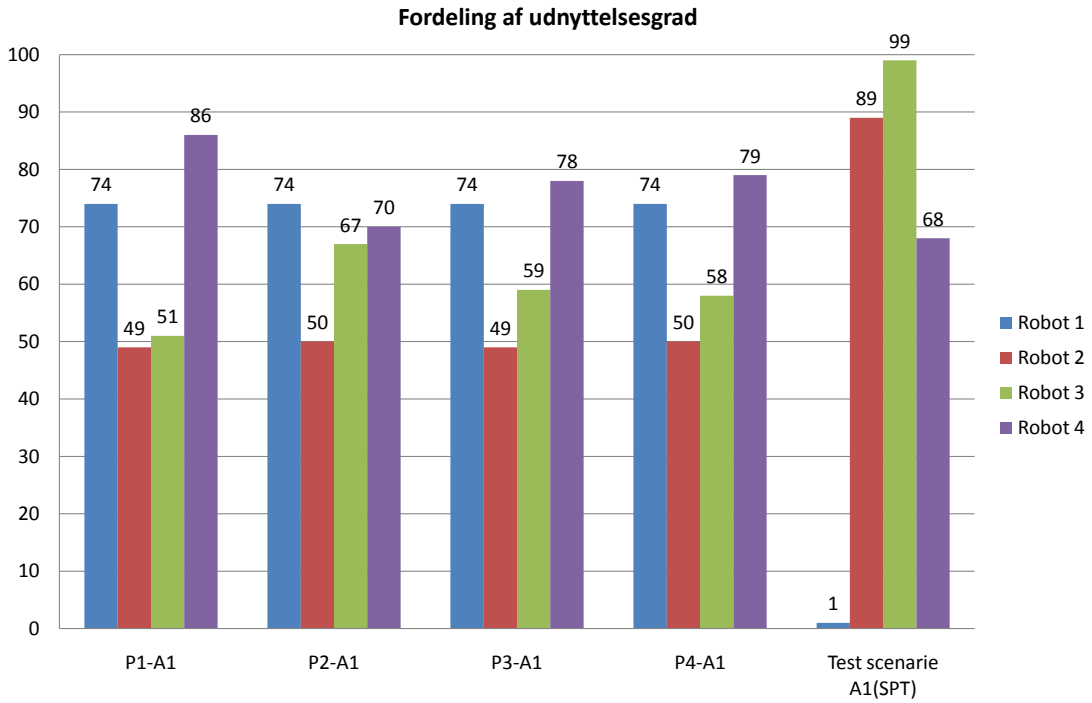


Figur 6.15: Udnyttelsesgrad for SPT fra Test scenarie A1.



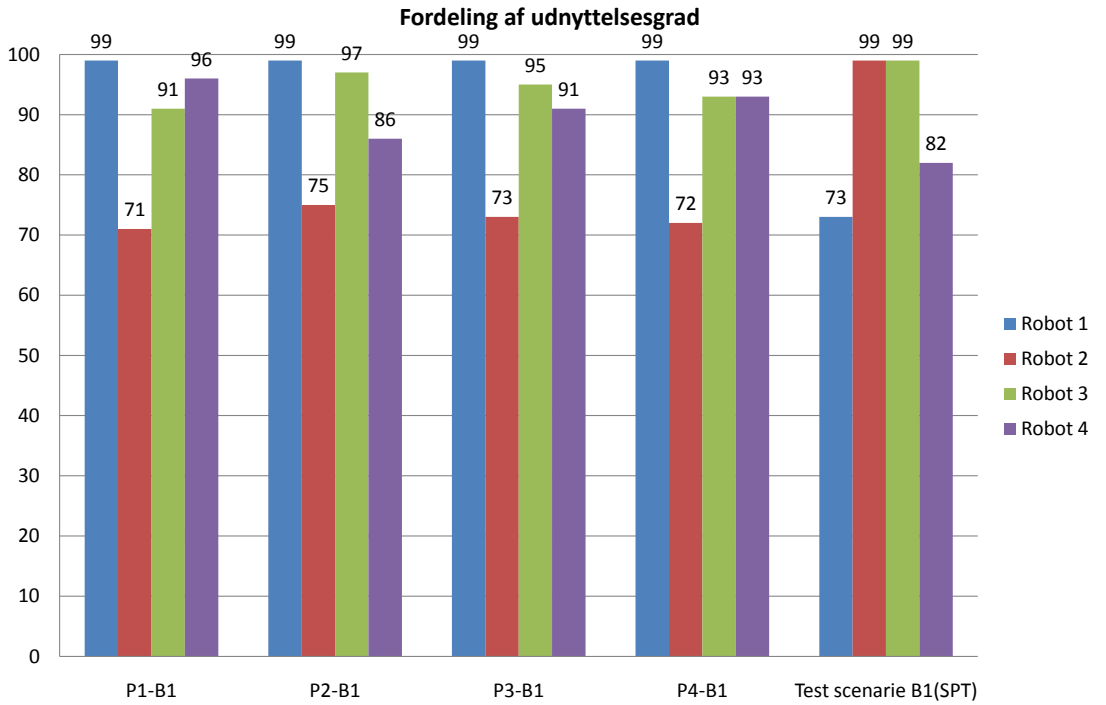
Figur 6.16: Udnyttelsesgrad for pakkestrategi P1, som har FIFO i robotcelle 4.

Det er meget interessant at se forskellen på udnyttelsesgraden for de enkelte robotter, og se hvordan pakkestrategierne fordeler arbejdet til robotterne, hvilket også kan ses på figur 6.17. På figuren er SPT fra Testfase 1 ved A1 medtaget, da den som nævnt minder meget om de nye pakkestrategier, og den gav nogle af de bedste resultater i Test fase 1.



Figur 6.17: Her ses robotternes udnyttelsesgrad for pakkestrategierne ved scenarie A1. Yderligere er forsøget fra Testfase 1 ved A1 med SPT medtaget. Bemærk den store forskel i, hvordan arbejdet bliver fordelt mellem robotterne.

Figuren viser at alle pakkestrategier stort set fordeler arbejdet på samme måde. Det er meget interessant at se udnyttelsesgraden for de enkelte robotter i SPT, hvor Robot 1 er helt nede på 1%. Dette ændrer de ny pakkestrategier, så Robot 1 kommer til at ligge på 74% i udnyttelsesgrad. Det er også interessant, at se hvordan fordeling bliver, når belastningen på pakkeanlægget stiger, hvilket vises på figur 6.18 på næste side for de nye pakkestrategier samt SPT ved belastning B1.

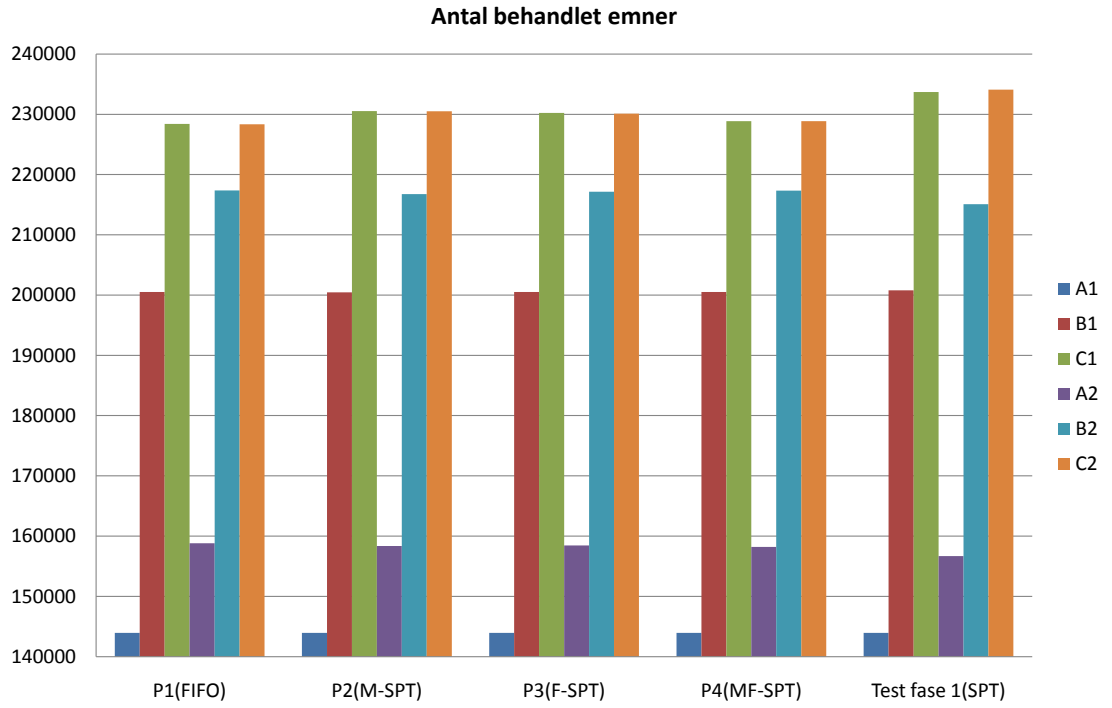


Figur 6.18: Her ses robotternes udnyttelsesgrad for pakkestrategierne og et forsøg fra Test scenarie B1 med SPT.

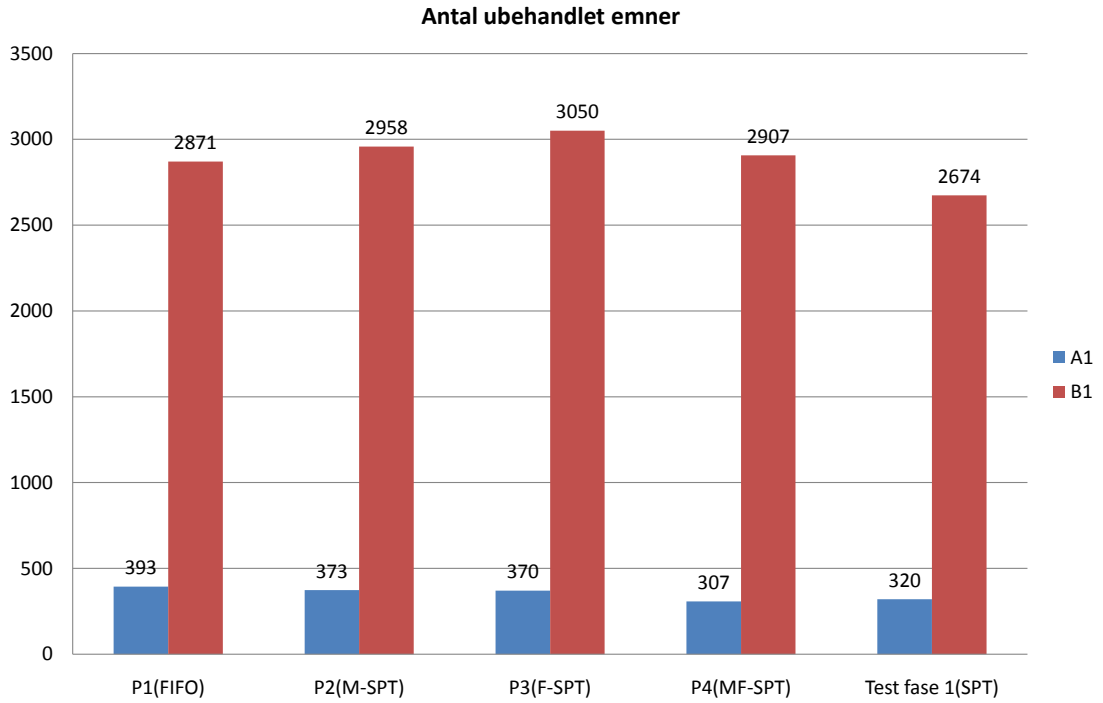
Det kan her observeres at den maksimale belastning på pakkeanlægget næsten er nået, da Robot 1,3 og 4 ligger over de 90% og derfor vises ikke en fordeling af udnyttelsesgraden for C1, da alle robotter ligger på 99%.

Fordelingen af udnyttelsesgraden er forskellig fra den tidligere Testfase 1 ved B1 med SPT. Her ligger Robot 2 og 3 på 99%, hvor de nye pakkestrategier har Robot 1 til at ligge på 99% og Robot 3 og 4 ligger omkring 90%. Dette skyldes reglen om minimum antal targets, som sørger for, at stort set alle kasser der ankommer til Robot 1, har 2 ledige targets. Dette, kombineret med at Robot 1 har så mange emner til rådighed, gør at den er konstant beskæftiget.

Der er altså ingen forskel i samlet udnyttelsesgrad mellem de nye pakkestrategier og SPT fra Testfase 1, men arbejdet bliver fordelt anerledes. Derfor synes det interessant, at se på antallet af behandlet og ubehandlet emner, da det vil give en identikation af om de nye pakkestrategier kan forøge antallet af behandlet emner eller nedsætte antallet af ubehandlet emner ifht. strategierne i Testfase 1. Se figur 6.19 og figur 6.20 på næste side.

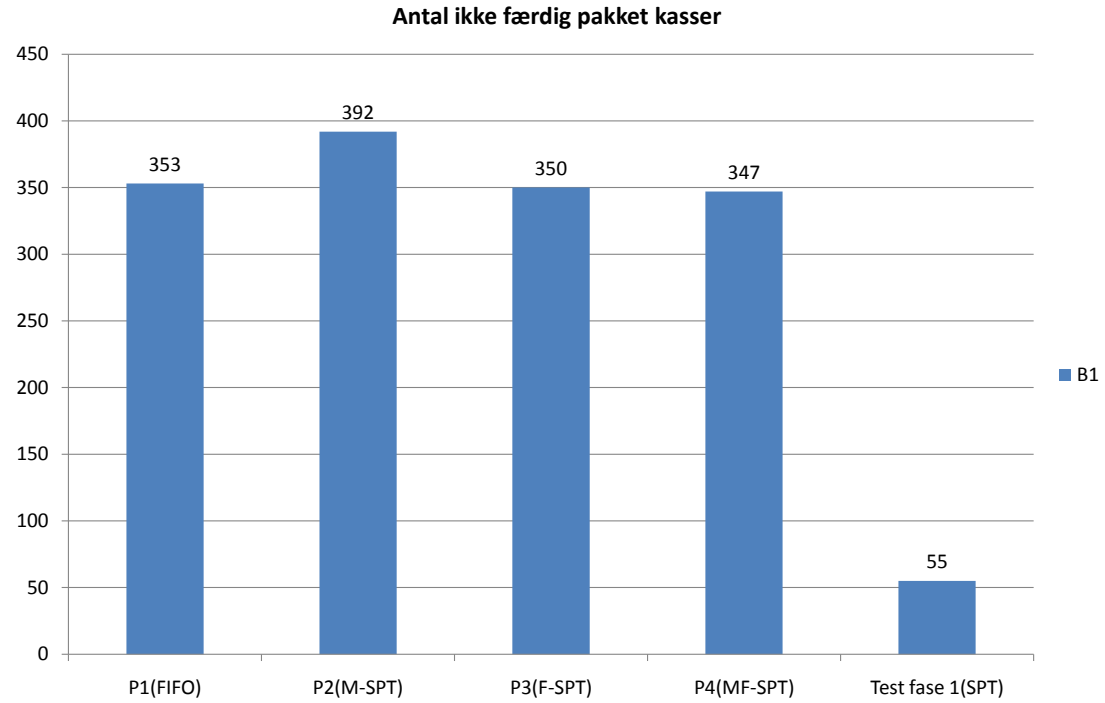


Figur 6.19: Her ses antallet af behandlet emner for de nye pakkestrategier samt for SPT fra Testfase 1.



Figur 6.20: Her ses antallet af ubehandlet emner for pakkestrategier og Test scenarie 1 ved hhv. A1 og B1 belastning.

Figuren viser at pakkestrategien P4 med MF-SPT i den sidste robotcelle har det laveste antal ubehandlet emner ved lav belastning(A1) ifht. de andre pakkestrategier. Det er dog kun i A1 at en af de nye pakkestrategier performer bedre end SPT fra Testfase 1, som er bedre når belastningen er mellem/høj på pakkeanlægget. Dette kan også ses på antallet af ikke færdig pakket kasser, som kan ses på figur 6.21 på den følgende side. Her er antallet 0 for alle pakkestrategier ved lav belastning (A1), men ved højere belastning, som i B1, er Testfase 1's SPT strategi omkring syv gange bedre end de nye pakkestrategier.

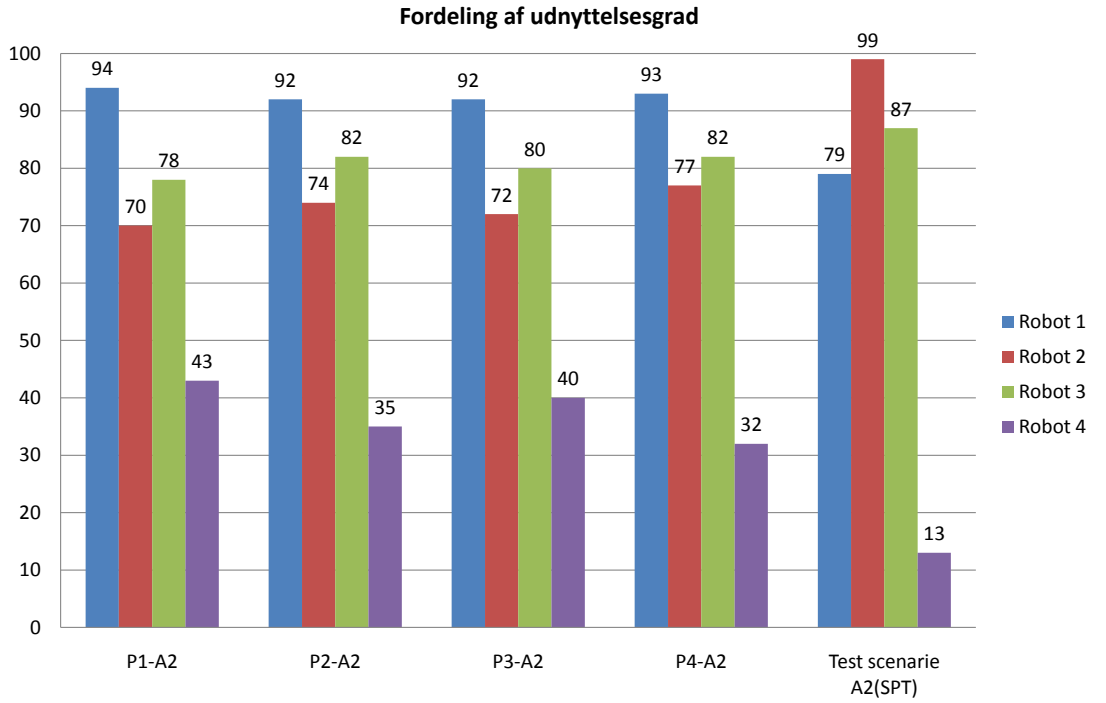


Figur 6.21: Her ses antallet af ikke færdig pakket kasser, hvor det viser sig at Testfase 1's SPT er 7 gange bedre.

Resultaterne ved at anvende de nye pakkestrategier ved A1, B1 og C1, synes derfor ikke at være bedre end dem fra Testfase 1's SPT. Derfor er det interessant at se om de nye pakkestrategier er bedre, når A2, B2 og C2 testes. Da Robot 2 og 3 var lavt belastet i Test scenarie 1, burde de blive bedre udnyttet, når de i Test scenarie 2 kan pakke 2. sorteringsemner.

6.2.2 Test scenarie 2

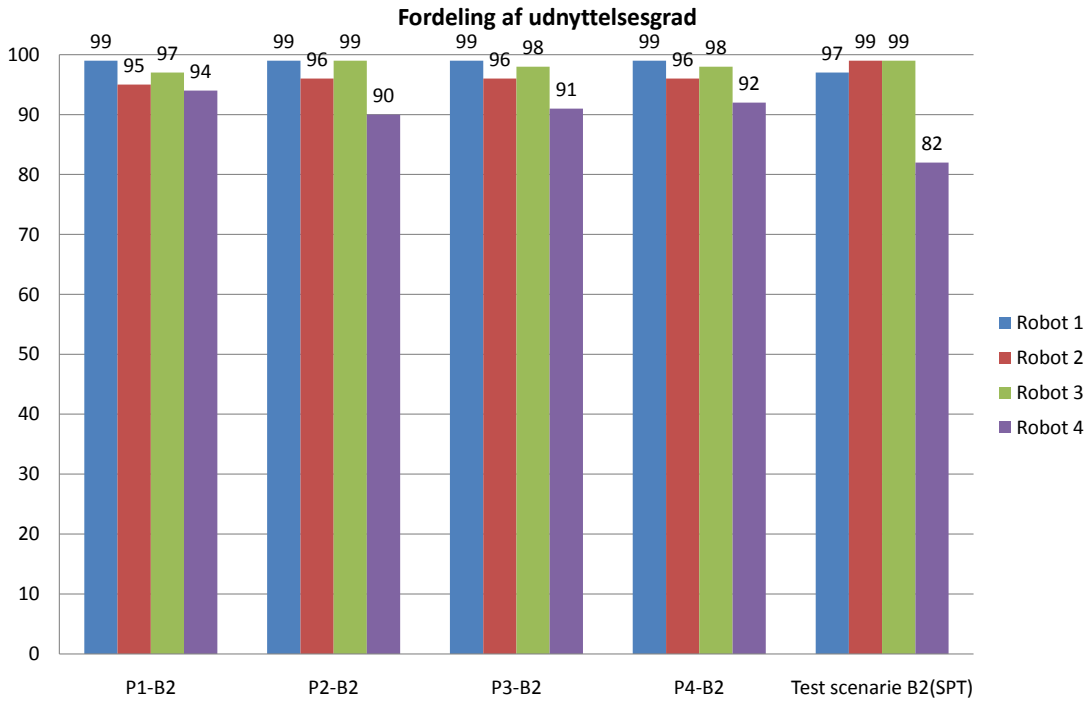
På figur 6.22 på næste side vises udnyttelsesgraden for de nye pakkestrategier.



Figur 6.22: Her ses udnyttelsesgraden for robotterne i scenarie A2, hvor alle nye pakkestrategierne og Testfase 1(SPT) er testet.

Det er tydeligt at udnyttelsesgraden for Robot 2 og 3 stiger i ifht. scenarie A1, som kan ses på figur 6.17 på side 98. I scenarie A1 har Robot 4 en meget højere udnyttelsesgrad end i dette tilfælde. Dette er helt klart betydningen af 2. sorteringsemner, hvor Robot 2 og 3 behandler flere af disse.

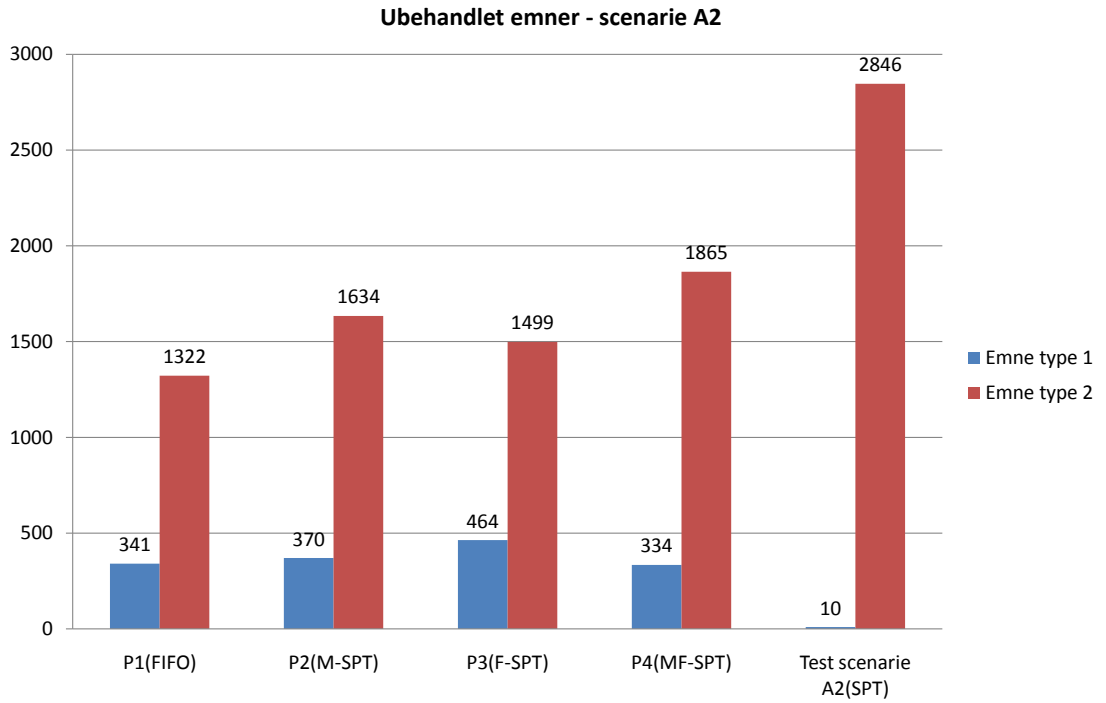
Udnyttelsesgraderne for Test scenarie B2 kan ses på figur 6.23 på næste side. Udnyttelsesgraden for Test scenarie C2 vises ikke, da alle robotterne har en 99% udnyttelsesgrad.



Figur 6.23: Her ses udnyttelsesgraden for robotterne i scenarie B2. Det skal bemærkes at de nye pakkestrategier gør at udnyttelsesgraden bliver mere homogen.

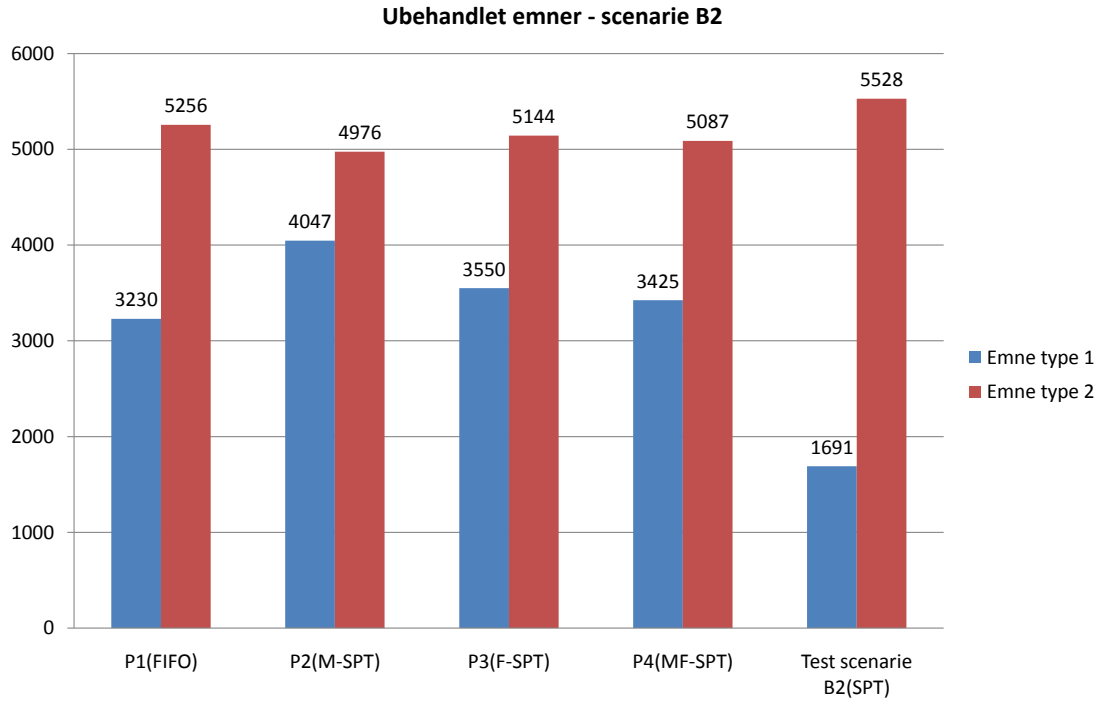
Som det ses på figurene, så er udnyttelsesgraderne ved A2 og B2 mere homogene i Testfase 2 end ved Testfase 1(SPT). Dette opfylder netop det kriterie, som tidligere blev opstillet, hvor det ønskes at formindske forskellen på udnyttelsesgraderne mellem robotterne. Dette er selvfølgelig positivt, men det må ikke gå ud over kriterierne om at sænke antallet af ubehandlet emner og ikke færdig pakket kasser.

På figur 6.24 på modstående side ses antallet af ubehandlet emner i Testfase 2 ved scenarie A2.



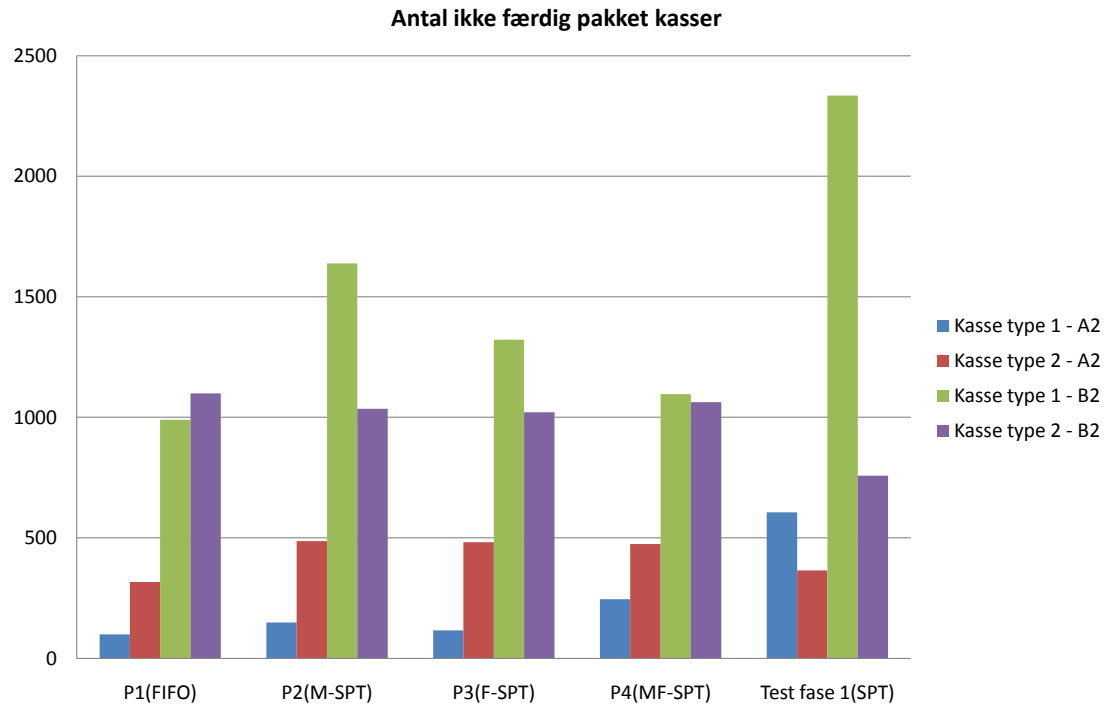
Figur 6.24: Her ses antallet af ubehandlet for scenarie A2 for strategierne i Testfase 2 og for SPT fra Testfase 1.

Der er meget stor forskel på antallet af ubehandlet emner i Testfase 2 og Testfase 1(SPT). I Testfase 1(SPT) pakkes næsten alle type 1 emnerne, men færre 2. sorteringsemner bliver pakket. Pakkestrategierne i Testfase 2 er altså dårligere til at pakke type 1 emner. Dette skyldes at pakkestrategierne i Testfase 2 bruger *minimum antal targets*, hvilket begrænser især robotcelle 2 og 3 fra at pakke type 1 emner, hvilket tvinger dem til at pakke en del type 2 emner. Den samme tendens vil vise sig ved højere belastning, som kan ses på figur 6.25 på næste side.



Figur 6.25: Her ses antallet af ubehandlet emner i scenarie B2.

Tendensen fra scenarie A2 går igen i scenarie B2. Pakkestrategierne performer næsten ens, hvor P1 har færrest ubehandlet type 1 emner og P2 har færrest ubehandlet 2. sorteringsemner. Det ønskes, at se nærmere på hvordan pakkestrategierne performer ifht. antallet af ikke færdig pakket kasser, se figur 6.26 på modstående side.



Figur 6.26: Antal ikke pakket kasser A2 og B2

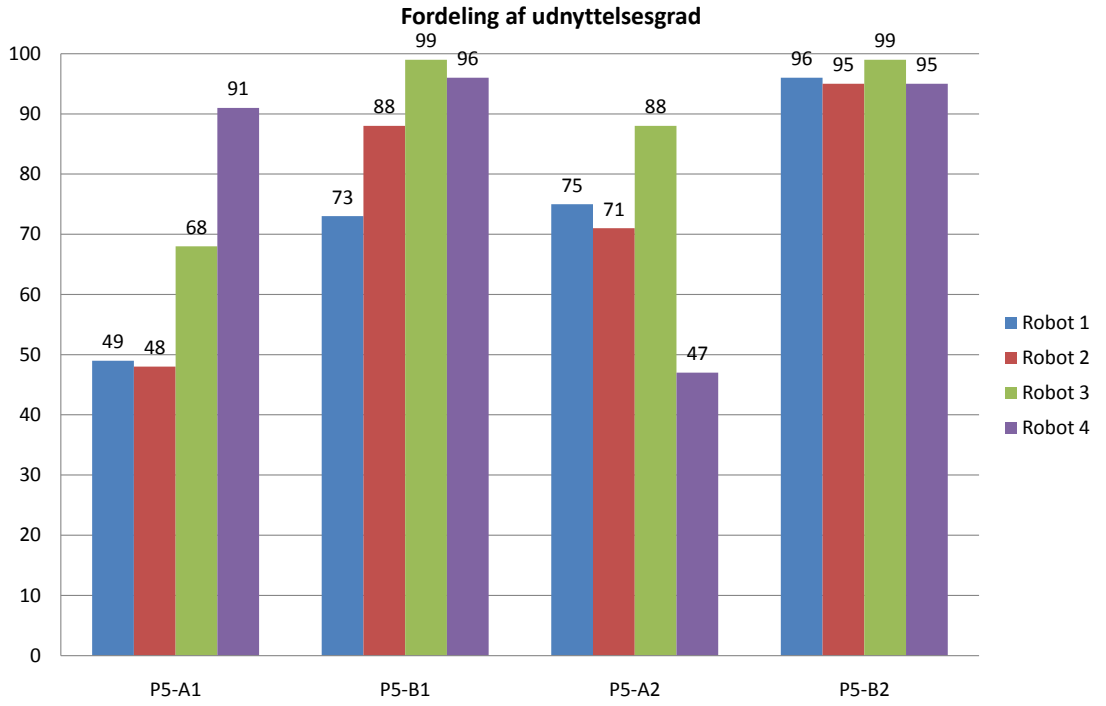
Det kan ses at alle pakkestrategierne ved lav belastning, altså A2, stort set har det samme antal ikke færdig pakket kasser. Det er først ved højere belastning, som i B2, at P1 og P4 er betydelige bedre end de to andre. En interessant observation er, at Testfase 1's SPT strategi performer meget dårligere mht. antallet af ikke færdig pakket kasser, hvilket ikke var tilfældet i Test scenarie 1, hvor Testfase 1's SPT strategi var 7 gange bedre end Testfase 2's pakkestrategier på dette område, se evt. figur 6.21 på side 102. Dette skyldes at i Testfase 2 er reglen om *minimum antal targets* blevet brugt - dette medfører, som tidligere nævnt, at stort set alle kasser ankommer til Robot 1 og mangler to emner, hvilket den nemt kan nå at fylde op. Denne jævne fordeling af emner i kasser gør, at disse strategier er mindre følsomme overfor den bølge af 2. sorteringsemner, der kommer engang i mellem. Det er netop ved disse bølger, at SPT pakkestrategien i Testfase 1, er følsom. Den har ingen jævn fordeling af emner i kasser - denne ujævnhed kombineret med en bølge af 2. sorteringsemner gør, at Robot 1 engang i mellem får en række type 1 kasser, som mangler for mange emner til, at den kan pakket dem. Det er altså en ret kompliceret situation, som kan udforskes meget mere end dette projekts tidsramme tillader.

Igennem alle testene synes pakkestrategi P1, som anvender FIFO i den sidste robotcelle, at være den bedste på de fleste områder. Derfor opstilles en modificeret pakkestrategi ud fra denne, som kaldes P5. P5's ændringer ihft. P1, bliver at der er vertikal prioritering type 1 i robotcelle 1 og begrænsningerne mht. antallet af minimum targets nedsættes. Pakkestrategi P5 kan ses i tabel 6.5.

	Robotcelle 1	**Robotcelle 2**	**Robotcelle 3**	**Robotcelle 4**
Prio. regel	SPT	SPT	SPT	FIFO
Vert. Prio.	Type 1	Type 1	Type 1	Ingen
Min. targets	0	1	2	0

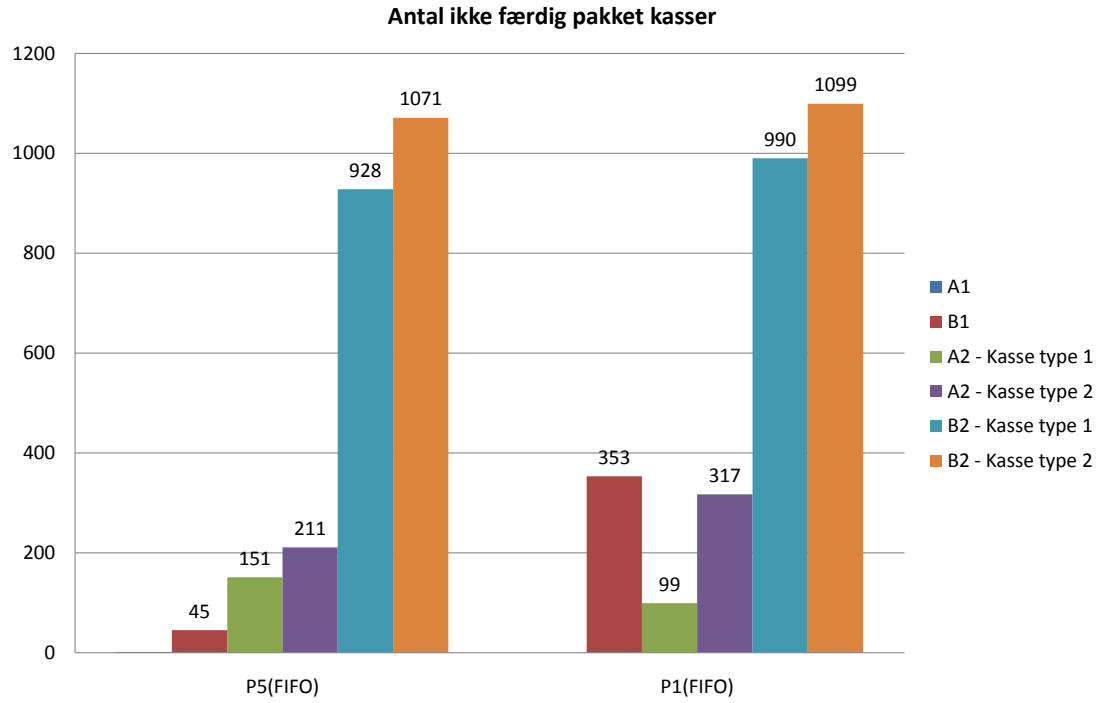
Tabel 6.5: Her ses indstillingerne for pakkestrategi P5.

Idéen med P5 er at bruge vertikal prioritering type 1 i de tre første robotceller for at sikre, at der til den sidste robotcelle kun er emner(type 1), som er tæt på kasserne - altså forsøges det at sikre, at Robot 4 får de hurtigste emner, så den bedre kan nå at pakke emnerne, inden de forlader anlægget. Dette burde betyde, at Robot 4 pakker væsentlig flere emner, og derfor sættes minimum antal targets ned for Robot 2 og 3, og fjernes totalt fra 4. Minimum antal targets er sat til 1 for Robot 2 - kasserne der ankommer til Robot 1 bør altså kun mangle et enkelt emne. Da Robot 1 pga. vertikal prioritering nu tager emner med høje procestider, er det netop vigigt at der er fjernet noget belastning fra denne.



Figur 6.27: Her ses udnyttelsesgraden for robotterne med pakkestrategi P5.

Det kan ses på figur 6.27 at fordelingen af belastningen er ændret ifht. scenarierne med pakkestrategi P1, da belastningen nu er koncentreret på Robot 3 og 4 - der er dog stadig stor forskel mellem det forskellige robotter, så udnyttelsesgraden er altså ikke blevet mere homogen.



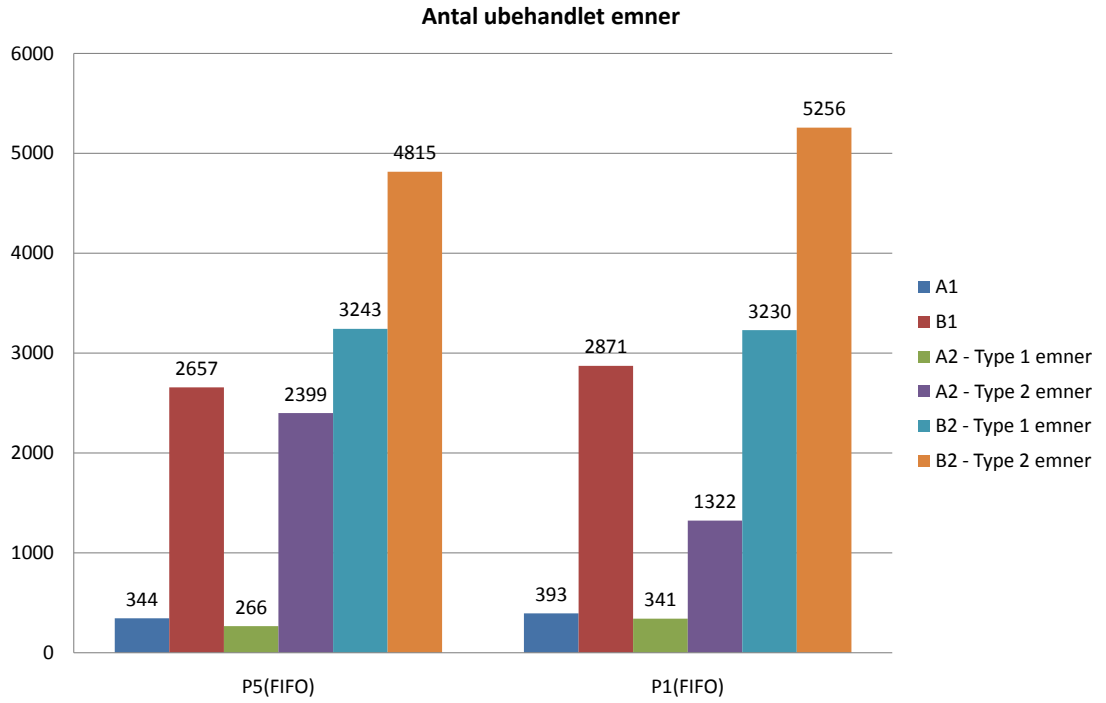
Figur 6.28: Antal ikke pakket kasser ved de lave belastninger for både Test scenarie 1 og 2 med pakkestrategi P5 og P1

Mht. ikke færdig pakket kasser performer P5 bedre i alle tests undtagen A2 kasse type 1. Dette er især i B1, hvor den formindsker antallet af ikke færdig pakket kasser fra 353 i P1 til 45. 45 er faktisk det laveste antal ikke færdig pakket kasser hidtil opnået i B1.

Kigges der på figur 6.29 på modstående side ses der ligeledes små forbedringer mht. antal ubehandlet emner med undtagelse af A2 type 2 emner.

P5 må derfor siges at performe på linje med end P1, med undtagelse af nogle få områder som er hhv. bedre og dårligere. P1 og P5 performer altså stort set ens, men belastningen fordeles anderledes i de to pakkestrategier.

P5 er den sidste pakkestrategi, som blev testet. Det lykkedes dermed aldrig at finde den perfekte pakkestrategi, men der kan stadig drages nogle interessante konklusioner ud fra forsøgene.



Figur 6.29: Antal ubehandlet emner ved de lave belastninger for både Test scenarie 1 og 2 med pakkestrategi P5 og P1

6.3 Konklusion af forsøgene

I dette afsnit opsamles resultaterne fra de forskellige Test scenarier og pakkestrategier og der konkluderes udfra disse, hvilke prioriteringsregler der kan anbefales til de forskellige scenarier.

Første testfase startede med Test scenarie 1, som er A1, B1 og C1, se evt. tabel 6.2 på side 78. Det kan i disse forsøg tydeligt ses at SPT-gruppen, altså SPT, M-SPT, F-SPT og MF-SPT, er de prioriteringsregler, som pakker flest emner i alle scenarierne. Yderligere er SPT-gruppen dem som har færrest ubehandlede emner, hvilket selvfølgelig hænger sammen med flest antal behandlet emner. Det skal dog nævnes at FIFO, ved lav belastning (A1), faktisk har et mindre antal ubehandlet emner end SPT og MF-SPT. Med hensyn til færrest antal ikke færdig pakket kasser, er det også SPT-gruppen, som er bedst ved B1 og C1 belastning, hvor især F-SPT synes at være en del bedre end de andre. FIFO er igen på samme niveau som SPT-gruppen ved lav belastning, og bedre end MF-SPT. Udfra dette test scenarie blev det udledt, at FIFO ville være en god idé at bruge som prioriteringsregel for kasserne i Robotcelle 1, samt for emnerne i Robotcelle

4.

MF-SPT synes ikke at fungere efter hensigten i dette test scenarie, hvilket måske skyldes at det optimale område, som er beskrevet i afsnit 2.4.1 på side 26, indskrænkes for meget og derved kun behandler emner inden for et lille område i arbejdsområdet.

I Test scenarie 2, som er A2, B2 og C2, blev type 2 emnerne inkluderet i forsøget. Belastningen på pakkeanlægget øges med 10%, da 10% af input emnerne er type 2 emner. Resultaterne viser igen at SPT-gruppen pakker flest emner, hvilket skyldes at disse stort set altid behandler emnet med den korteste procestid, og dermed kan nå at pakke flest emner. En interessant observation er at udnyttelsesgraden på hver enkelt robot ændre sig markant ifht. Test scenarie 1. Her har Robot 1 stort set intet at lave, hvilket skyldes at alle kasserne er færdig pakket, når de ankommer til robotcelle 1. I Test scenarie 2 stiger udnyttelsesgraden for Robot 1, hvilket skyldes 2. sorteringsemnerne, som denne kan behandle, når der ikke er plads i type 1 kasserne til at pakke type 1 emner.

En interessant observation blev her gjort. Det blev opdaget hvordan Robot 1 og 4 "kæmper" direkte mod hinanden om at pakke emner ved lav belastning. "Bølger" af type 2 emner forstyrrede rytmen og var direkte skyld i, at belastningen skiftede mellem de to robotter. Hvis man forskede videre i dette område, kunne det måske blive muligt at genkende disse bølger ved deres ankomst og reagere herpå således, at fordelingen af arbejde blev vedligeholdt.

Mht. antallet af ubehandlet emner er FIFO igen god ved lav belastning, hvor SPT dog er bedst til at pakke type 1 emner, men har det højeste antal ubehandlet type 2 emner. Antallet af ikke færdig pakket kasser er SPT-gruppen igen bedst, hvor især F-SPT er bedre end de andre ved høj belastning.

I Testfase 2 blev nogle nye pakkestrategier kombineret ud fra alle tre pakkeregler. I P1, P2, P3 og P4 er der indført vertikal prioritering og der er sat et minimum antal targets, som skal være tilbage i kasserne, når de forlader et arbejdsområde. Disse indstillinger er ens i robotcelle 1,2 og 3, som er med prioriteringsregel SPT, og i robotcelle 4 er prioriteringsreglen hhv. FIFO, M-SPT, F-SPT og MF-SPT. Disse er valgt på baggrund af resultaterne i Testfase 1. Indførslen af vertikal prioritering skulle gerne medføre, at sidste robotcelle fik emner med lavere procestider, og dermed bedre kunne nå at pakke dem, inden de kørte ud af anlægget. Minimum antal targets skulle gerne være med til, at fordele arbejdet bedre ved lav belastning, samt jævne antallet af emner pakket i type 1 kasserne, når de ankomt til Robot 1.

Resultatet var at fordelingen blev stort set ens i alle pakkestrategier, og den største

ændring fra Test scenarie 1 og 2, er at Robot 1 stiger fra 1% til 74% udnyttelsesgrad - desuden falder udnyttelsesgraden for Robot 2 kraftigt. Alt i alt er arbejdsfordelingen blevet mere lige, dog langt fra homogen.

Mht. antallet af behandlet emner og antallet af ubehandlet emner, er alle pakkestrategierne stort set lige gode i alle scenarierne. P1 med FIFO, synes dog at være lidt bedre mht. ubehandlet type 1 emner og kasser.

Slutteligt blev en sidste kombination prøvet af, P5. Denne performede på linje med P1, med et par enkelt resultater som var henholdsvis dårligere og bedre. Fordelingen af arbejdet blev dog markant anerledes, en konsekvens af at der blev brugt et andet minimum antal targets.

Alt i alt har disse forsøg givet en identifikation af hvilke prioriteringsregler og pakkestrategier, som er bedst til at opfylde de tidligere stillede kriterier, som ses igen i tabel 6.6.

Minimeres	**Maksimeres**
-	- Den homogene fordeling af arbejdsbelastningen (P2)
-	- Udnyttelsesgraden på pakkeanlægget (LIFO)
- Antal ubehandlet emner (SPT)	- Antal behandlet emner (SPT)
- Antal ikke færdig pakket kasser (P1)	- Antal pakket kasser (P1)

Tabel 6.6: Her ses de kriterier som ønskes minimeres og maksimeres.

Dette giver et helt klart billede af, at SPT generelt er en god prioriteringsregel, når der gælder om at få pakket flest mulige emner. Pakkestrategien P1 var generelt den bedste til at minimere antallet af ikke færdig pakket kasser og dermed også antal pakket kasser. LIFO giver den højeste udnyttelsesgrad for pakkeanlægget, dog kun pga. lange procestider - LIFO anbefales ikke.

Minimum antal targets har vist sig som et stærkt værktøj til at fordele arbejdet. Dog har det voldsomme effekter, hver gang man ændrer et minimum med bare et enkelt emne. En konklusion må derfor være, at denne regel kan forbedres. En mulighed kunne være at i stedet for et minimum antal targets, var det en procentdel af targets. Det ville betyde, at pakkestrategien selv skulle sørge for at overholde procentdelen, og ville skifte mellem f.eks. 2 og 3 antal frie targets. Dette ville højst sandsynligt være med til at gøre reglen mere præcis til at fordele arbejdet mellem robotterne.

Ud fra resultaterne kan kombineres en strategi, P6, som tager de bedste elementer fra forsøgene, se tabel 6.7 på den følgende side. Det anbefales at teste denne strategi.

	Robotcelle 1	Robotcelle 2	Robotcelle 3	Robotcelle 4
Prio. regel - emner	SPT	SPT	SPT	FIFO
Prio. regel - kasser	FIFO	SPT	SPT	SPT
Vert. Prio.	Ingen	Type 1	Type 1	Ingen
Min. targets	0	1-2	2-3	0

Tabel 6.7: Denne pakkestrategi, P6, anbefales som følge af forsøgene. Bemærk at minimum antal targets ikke ligger fast, det bør testes, hvilken der er bedst.

Alle forsøgene har dermed givet en bedre forståelse for, hvorledes pakkestrategierne performer ifht. hinanden. Resultaterne er helt klart ikke optimale og kan godt forbedres ved at lave yderligere forsøg. Simuleringen videregives til OCTOMATION, som får et stærkt værktøj til at afprøve yderligere forsøg og forhåbentlig finde en pakkestrategi de evt. kan implementere i deres pakkeanlæg og opnå en forbedret ydelse.

7 Konklusion

Det er lykkedes at udvikle et brugervenligt og fleksibelt simuleringsværktøj, som kan simulere et pakkeanlæg bestående af 1-8 robotceller, 2-3 transportbånd (1 til emner, 1-2 til kasser) og er programmeret med C#. Værktøjet giver nøgletal for hvordan anlægget performer, og gør det muligt at udvikle og evaluere nye pakkestrategier.

Simuleringsværktøjet består af en brugerflade, hvor alle indstillinger som ønskes i simuleringen opsættes i Hovedmenuen. I Hovedmenuen er der anvendt generelle designregler, hvilket har medført et brugervenligt og overskueligt vindue. Der bliver ikke præsenteret mere end 7 ± 2 informationer ad gangen, hvilket gør at brugeren ikke mister overblikket. Yderligere er der lagt fokus på at have kontinuitet i Hovedmenuen, således alle undervinduer og faner har samme udformning. De vigtigste indstillinger i Hovedmenuen er input raten af emner og kasser, hastigheden for transportbåndene, antal robotceller, pakkestrategier, ændre dimensioner for pakkeanlægget og opsætte slutbetingelser for simuleringen. Disse indstillinger giver brugeren mulighed for at lave forsøg med mange forskellige anlæg og arbejdsbelastninger.

Simuleringsværktøjet gør det muligt for brugeren at sammensætte pakkestrategier ud fra de tre regler om *heuristiske prioriteringsregler*, *vertikal prioritering* og *minimum antal targets*. De heuristiske prioriteringsregler kan vælges for hele anlægget, eller specifikt vælges for hver enkelt robotcelle, både for emner og kasser.

Alle disse muligheder for opsætning af pakkeanlæg, arbejdsbelastning og pakkestrategi gør simuleringsværktøjet fleksibelt.

Selve simuleringen foregår i Simuleringsvinduet, hvor pakkeanlægget ses ovenfra og alle komponenterne, såsom transportbånd, emner, kasser samt robotternes End-effector bliver vist. Dette gør det muligt at følge med og vurdere, hvorledes robotterne pakker ifht. den valgte pakkestrategi. Yderligere er der lavet et statistik område, hvor der vises nøgletal og søjlediagrammer, hvilket gør det overskueligt, at se hvordan pakkeanlægget performer under simuleringen. Der er også anvendt nogle generelle designregler i Simuleringsvinduet, som bl.a. er farvevalg. Her kan der nævnes at kasser, som ikke er færdig pakket og forlader pakkeanlægget, markes med rød for signalere fejl.

Hele simuleringsværktøjet er verificeret og valideret. Verificeringen viser, at de kravene til simuleringsværktøjet er opfyldt. Der er kun ét krav som ikke er opfyldt, hvilket er tilkoblingen af Octomations Statistik modul, som ikke kunne anskaffes inden for projektets tidsramme. I stedet er der udarbejdet et statistik modul i MATLAB, som kan

analysere resultaterne fra en simulering. Valideringen er gjort i to faser, test af komponenter og test af hele programmet. Komponenterne blev testet hver for sig og valideret til at kunne udføre de funktioner, som tilsammen udgøre simuleringsværktøjet. I test af hele programmet, er det observeret, at sammenspillet mellem komponenterne fungerer korrekt og giver ikke nogle fejl. Dette kan f.eks. ses ved, at robottens end-effector opsamler det rigtige emne og kommer det i den korrekte kasse, alt efter hvilken prioriteringsregel som er valgt.

Der blev med simuleringsværktøjet foretaget en række forsøg med i alt 12 forskellige pakkestrategier. De blev alle testet i to forskellige scenarier, som hver havde tre belastningsniveauer. Resultaterne gav ikke noget entydigt svar på, hvilken pakkestrategi der var bedst. Pakkestrategien P1 skal dog fremhæves, da den var bedst til at minimere antallet af ikke færdig pakket kasser og dermed også antal pakket kasser. Yderligere blev det konkluderet, at FIFO ville være en god idé at bruge som prioriteringsregel for kasserne i Robotcelle 1. Resultaterne fra Test fase 2 viste at *Minimal antal target*-reglen er et stærkt værktøj til at fordele arbejdsbelastningen. Resultaterne fra forsøgene førte alt i alt frem til en pakkestrategi, P6, som sandsynligvis vil performe bedre end de afprøvede pakkestrategier og bør testes nærmere.

Hermed kan det konkluderes, at det er lykkedes at udvikle et fleksibelt og brugervenligt simuleringsværktøj, som kan anvendes til afprøvning og vurdering af pakkestrategier. Simuleringsværktøjet lever op til de stillede krav og OCTOMATION kan med dette simuleringsværktøj afprøve forskellige pakkestrategier uden at bruge mange ressourcer på forsøg i værkstedet.

8 Videre arbejde

I dette afsnit præsenteres en handlingsplan for hvilke forbedringer, der kan laves i octoSimulation. Yderligere anbefales der en pakkestrategi, som bør afprøves i simuleringen. Dette gøres for at give OCTOMATION nogle områder, de kan arbejde videre med.

I simuleringen bør der indføres en model af robottens kinematik og dynamik, samt en metode som regner robottens korrekte ruter ud, hvilket ikke sker i den nuværende simulering. Udfra ruten kan modellen regne en mere præcis procestid ud.

Et andet punkt er at tilkoble OCTOMATION statistik modul, som skal analysere resultaterne fra simuleringerne. Disse resultater udskrives pt. i txt filer, men burde skrives ind i OCTOMATIONs database, ligesom det sker med pakkeanlæggene ude ved kunderne. Yderligere kan det være fordelagtig at anvende OCTOMATION statistik modul under selve simuleringen.

Minimal antal targets er et stærkt værktøj til at fordele arbejdsbelastningen, men påvirker pt. fordelingen meget kraftigt, når indstillingen af reglen bliver justeret med bare et enkelt emne. Det anbefales derfor, at denne regel bliver videreudviklet til f.eks. at være procentdel af antal targets i kassen. Så ville brugeren kunne justere arbejdsfordelingen mere præcist.

Det anbefales også at udføre flere test. En pakkestrategi som sandsynligvis vil minimere antallet af ikke færdig pakket kasser kan ses i tabel 8.1. Denne pakkestrategi skal gerne minimere antallet af ikke færdig pakket kasser type 1, fordi der anvendes FIFO i robotcelle 1. Dette vil medføre, at robotten opsamler emner tæt på den øvre grænse og kommer dem i den første kasse, som er ankommet i robottens arbejdsområde. Denne kasse er tættest på at forlade pakkeanlægget (på grund af crossflowet), og dermed pakkes altid et emne i denne kasse. Dermed vil kasserne altid blive færdig pakket, hvis de på dette tidspunkt kun mangler ét emne for at være færdig pakket.

	Robotcelle 1	**Robotcelle 2**	**Robotcelle 3**	**Robotcelle 4**
Prio. regel - emner	SPT	SPT	SPT	FIFO
Prio. regel - kasser	FIFO	SPT	SPT	SPT
Vert. Prio.	Ingen	Type 1	Type 1	Ingen
Min. targets	0	1-2	2-3	0

Tabel 8.1: Denne pakkestrategi, P6, anbefales som følge af forsøgene. Bemærk at minimum antal targets ikke ligger fast, det bør testes, hvilken der er bedst.

Litteratur

Adept Technology Inc. http://www.adept.com, 2011.

Shah Newaz Alam. http://www.buzzle.com/articles/waterfall-model-advantages-and-disadvantages.html, 2011.

Donald Bell. Uml basic: The sequence diagram. UML basic - The sequence diagram.pdf, 2004.

J. A. Bondy and U. S. R. Murty. *Graph Theory*. Springer, 2008.

Dassault Systemes. http://www.3ds.com/products/delmia/welcome/, 2011.

Persi Diaconis. The problem of thinking too much. The Problem of Thinking Too Much.pdf, 2003.

Benny Endelt. Gradient based optimization schemes. Gradient Based Optimization Schemes.pdf, 2009.

InControl. http://www.incontrolsim.com/, 2011.

InMóTx. Octomation naming convention. Naming a Octocell.pdf, 2010.

Lars Mathiassen, Andreas Munk-Madsen, Peter Axel Nielsen, and Jan Stage. Objekt orienteret analyse og design. Objekt Orienteret Analyse og Design.pdf, 2001.

R. Mattone, M. Divona, and A. Wolf. Sorting of items on a moving conveyor belt. Sorting of items on a moving conveyor belt.pdf, 1999.

MSDN. http://msdn.microsoft.com/en-us/library/system.random.aspx, 2011.

Peter Nielsen. Informationer fra møde afholdt d. 5. maj, 2011.

Naomi Nishimura. Pseudocode. Pseudocode.pdf, 2011.

OCTOMATION. `http://www.inmotx.com`, 2011.

Michael L. Pinedo. *Scheduling - Theory, Algorithms, and Systems*. Springer, 3 edition, 2008.

RAHBEK. `http://www.rahbek.dk/`, 2011.

RobWork. `http://www.robwork.dk/jrobwork/`, 2011.

Ian Sommerville. *Software Engineering*. Addison-Wesley, 8 edition, 2007.

William J. Stevenson. *Operations Management*. McGraw-Hill/Irwin, 9 edition, 2007.

Xcelgo A/S. `http://xcelgo.com/`, 2011.

MATLAB statistik modul

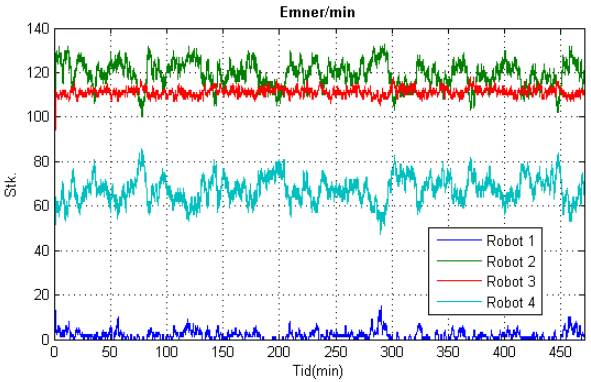
A

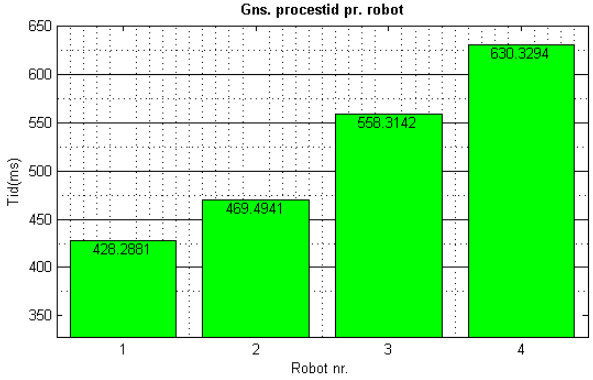
Til at analysere data efter en simulering er færdig er der udarbejdet et statistik modul i MATLAB, som ses på figur A.1 på modstående side. Formålet med dette statistik modulet er, at kunne gemme resultaterne fra en simulering på en overskueligt måde. Dette gør det nemmere at sammenligne resultater for forskellige simuleringer. I statistik modulet vises alle nøgletal og grafer. Graferne som vises er Emner/min, Udnyttelsesgrad pr. robot og den gennemsnitlig procestid for robotterne. Alle grafer vises for hele simuleringsforløbet. MATLAB statistik modulet kan findes på CD'en, hvor det dog kræver MATLAB for at benytte dette.

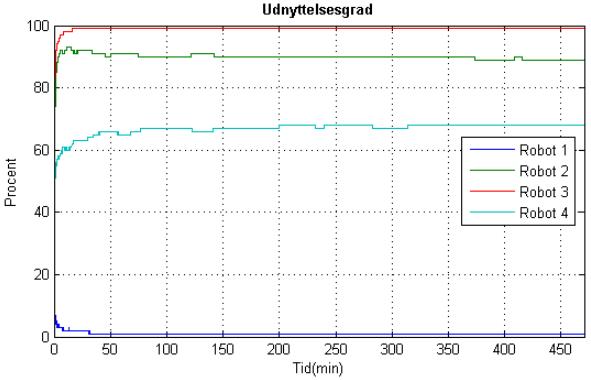
Resultater:

Behandlet emner:	143956
Gns. antal Emner/min:	299
Udnyttelsesgrad:	64
Ubehandlet emner:	320
Pakket kasser:	23983
Ikke-færdig pakket kasser:	0

Emne og kasse type 1:		**Emne og kasse type 2:**	
Behandlet emner:	143956	Behandlet emner:	0
Ubehandlet emner:	320	Ubehandlet emner:	0
Pakket kasser:	23983	Pakket kasser:	0
Ikke-færdig pakket kasser:	0	Ikke-færdig pakket kasser:	0







Figur A.1: Her ses MATLAB statistik modulet med nøgletal og grafer. Det er Test scenarie A1 for SPT som er vist.