Partial differential equations in modelling and control of manufacturing systems

R.A. van den Berg

SE 420379

Master's Thesis

Supervisor: Prof.dr.ir. J.E. Rooda
Coach: Dr.ir. A.A.J. Lefeber

Eindhoven University of Technology
Department of Mechanical Engineering
Systems Engineering Group

Eindhoven, March 2004

Preface

With the graduation project that is described in this thesis, I have reached the end of the five year curriculum of Mechanical Engineering at the Eindhoven University of Technology. During these years, I amassed a lot of knowledge in the fields of mathematics, physics, dynamics, and control science and I learned to think 'the academic way'. After three years of study, I joined the Systems Engineering Group, which is one of the nine research groups at the Department of Mechanical Engineering. Here, I was offered whole new challenges, such as the internship in France and the participation in a control conference, from which I gained much experience. The group has also provided me with this graduation project, which has been a challenging piece of research.

The project took place within the 'Control and Optimization' research theme of the Systems Engineering Group. The research in this theme is concerned with the development of controller design and optimization tools for manufacturing systems and machines. My research has been concerned with the exploration of a new technique for simulation and control of manufacturing systems. This technique concerns the use of partial differential equations (PDEs) in the modelling of these systems. Since this research topic is still in an early stage of development, this thesis is written in an 'introductory' way. It is meant as a starting point for other people who wish to continue the research in this field.

During my graduation project, I was supported by several people, whom I would like to thank here. First of all, my gratitude goes to my coach, Erjen Lefeber, for his constructive support and pleasant cooperation. Furthermore, I thank professor Rooda, who offered me a lot of opportunities during the past years at the Systems Engineering Group. Another word of thanks goes to professor Armbruster of Arizona State University for his helpful cooperation. Professor Armbruster performs similar research on PDE-models and his knowledge has been a valuable contribution to my research. I also thank my fellow students at the Systems Engineering Group for their delay support and the pleasant working environment they created. A special word of gratitude goes to Ad Kock and Joost van Eekelen for reviewing this thesis. Finally, I thank my family, friends and last but not least my girlfriend Willemijn for their patient support during my study.

Roel van den Berg
Eindhoven, March 2004

Summary

Due to the ever increasing complexity of manufacturing systems, the simple heuristics and operator experience that are currently used to control these systems, will at a certain point become insufficient. Therefore, in this research, a recently developed control framework is considered, which aims to control manufacturing systems using a continuous controller. The framework consists of three steps: the design of a continuous approximation model for the manufacturing system to be controlled, the design of a controller for this approximation model, and the coupling of the resulting controller to the manufacturing system by means of two converters.

For the design of the approximation model, two demands should be taken into account. Firstly, the characteristic system parameters, i.e., the throughput and the flow time, should be accurately described in both transient and steady state. Secondly, the approximation model should be suitable for the design of a continuous controller, which implies that the model must be continuous and computationally feasible. None of the available approximation models for manufacturing systems, however, complies with all these demands, and therefore a new class of models, namely PDE-models (PDE: partial differential equation), is considered. These models are continuous and computationally feasible, but their description of a manufacturing system's behaviour has not been validated so far. In this report, therefore, the suitability of PDE-models in simulation of manufacturing systems is investigated. A second aim of this research project is to investigate the possibilities of such a PDE-model in the control of a manufacturing system, when using the framework described above.

Using a discrete event model (DEM) for validation, the performance of three available PDE-models has been examined. The case that was used for this validation study is a flow line, consisting of identical M/M/1 processes. With this flow line, experiments have been performed that include both ramp up and ramp down simulations. The results of this validation study show that the PDE-models accurately describe the steady state behaviour of a manufacturing system, but that the description of the transient behaviour leaves much to be desired. In addition to this, one of the PDE-models shows undesirable behaviour: lots that enter the system influence the velocity of lots further downstream in the system. The other two PDE-models do not show this behaviour and are comparable in their overall performance. Of these two models, the most simple model is selected for use in the control framework.

For the control of the selected PDE-model, several control methods have been investigated. Only boundary control methods were considered, since the arrival rate of the manufacturing system (the influx for the PDE-model) is in this research assumed to be the only controllable variable. The two most promising control strategies, Lyapunov's stability theory and nonlinear model predictive control (nl-MPC), have been investigated further using three control problems that are relevant for the manufacturing industry. These problems include time optimal switching between steady states (i.e., ramp up or ramp down) and preventing permanent backlogs. With nl-MPC, a solution was found numerically for all the considered control problems. Lyapunov's stability theory, on the other hand, could only be applied to two of the control problems. Nevertheless, it did provide an analytical solution for the time optimal switching between steady states. The corresponding controller is often used in practice. The problem of preventing a permanent backlog (due to a switch between steady states) was solved analytically as well.

For the last step in the control framework, i.e., the coupling of the continuous controller to the manufacturing system (here represented by a DEM), two converters have been designed: an input converter, which converts the continuous output of the controller into a suitable input signal for the DEM, and an output converter, which converts the output of the DEM into suitable data for the controller. The input converter is simple: the influx as determined by the controller is used as the mean of the exponentially distributed arrival rate. For the output converter, a first order non-model based observer is used, which filters out the disturbance (stochastic behaviour) of the DEM's output and provides mean values to the controller. With these converters, the nl-MPC controller is successfully coupled to the DEM.

Samenvatting (in Dutch)

Doordat de complexiteit van fabricagesystemen blijft toenemen, zal op een gegeven moment het punt bereikt worden waarop de eenvoudige heuristieken en de ervaring van operators, die momenteel gebruikt worden om deze systemen te regelen, niet meer voldoende zijn. In dit onderzoek wordt daarom een relatief nieuw regelconcept beschouwd, dat tot doel heeft om fabricagesystemen te regelen met een continue regelaar. Het concept is opgebouwd uit drie stappen: het ontwerpen van een continu benaderingsmodel voor het te regelen fabricagesysteem, het ontwerpen van een regelaar voor dit benaderingsmodel en het koppelen van de resulterende regelaar aan het fabricagesysteem met behulp van twee convertors.

Aan het ontwerp van het benaderingsmodel worden twee eisen gesteld. Allereerst moeten de karakteristieke systeemparameters (de doorzet en de doorlooptijd) nauwkeurig beschreven worden, zowel in een overgangstoestand als in een evenwichtstoestand. De tweede eis is dat er voor het benaderingsmodel een continue regelaar ontworpen moet kunnen worden. Dit houdt in dat het benaderingsmodel continu moet zijn en binnen beperkte tijd doorgerekend moet kunnen worden. Geen van de bestaande benaderingsmodellen voor fabricagesystemen voldoet echter aan beide eisen. Daarom is er gekeken naar een nieuwe klasse van modellen: PDE-modellen (PDE is de Engelse afkorting voor partiële differentiaalvergelijking). Deze modellen zijn continu en kunnen snel doorgerekend worden. Of het gedrag van een fabricagesysteem door deze modellen nauwkeurig wordt beschreven, is echter nog nooit gevalideerd. In dit verslag is daarom onderzocht of PDE-modellen geschikt zijn voor simulatie van fabricagesystemen. Verder is onderzocht wat de mogelijkheden van een dergelijk PDE-model zijn bij het regelen van een fabricagesysteem, indien er gebruikt wordt gemaakt van het eerder beschreven regelconcept.

De prestatie van drie beschikbare PDE-modellen is onderzocht. Hierbij is een discrete-event model (DEM) gebruikt als validatiemodel. Het systeem dat gebruikt is voor deze validatie is een fabricagelijn, bestaande uit identieke M/M/1 processen. De experimenten die met deze fabricagelijn zijn uitgevoerd, bevatten zowel simulaties die een toename van de evenwichts-doorzet beschrijven als simulaties die een afname van de evenwichts-doorzet beschrijven. De resultaten van deze validatiestudie wijzen uit dat de beschouwde PDE-modellen de evenwichtstoestanden nauwkeurig beschrijven, maar dat de beschrijving van de overgangstoestand veel te wensen over laat. Bovendien ver-

toont één van de PDE-modellen ongewenst gedrag: lots die het systeem binnenkomen beïnvloeden de snelheid van lots die zich verderop in het systeem bevinden. De andere twee PDE-modellen vertonen dit ongewenste gedrag niet en zijn qua totaalprestatie vergelijkbaar. Het eenvoudigste model van deze twee is geselecteerd om gebruikt te worden in het regelconcept.

Voor het regelen van het geselecteerde PDE-model zijn verschillende regelmethodes onderzocht. Echter, alleen regelmethodes die het PDE-model via de randvoorwaarden kunnen regelen zijn beschouwd, aangezien in dit onderzoek wordt aangenomen dat de invoersnelheid van lots in het fabricagesysteem (de ingaande flux voor het PDE-model) de enige regelbare parameter is. De twee meest belovende regelstrategieën, Lyapunov's stabiliteitstheorie en niet-lineaire 'model predictive control' (nl-MPC), zijn nader onderzocht aan de hand van drie regelproblemen die relevant zijn voor fabricagesystemen. Deze problemen bevatten onder andere het stabiliseren van een systeem in een nieuwe evenwichtstoestand (zowel hogere als lagere evenwichts-doorzet) in de kortst mogelijke tijd en het voorkomen van een permanent productietekort (of -overschot). Met behulp van nl-MPC zijn er op numerieke wijze oplossingen gevonden voor alle beschouwde regelproblemen. Lyapunov's stabiliteitstheorie daarentegen, kon slechts voor twee van deze regelproblemen worden toegepast, maar heeft wel geresulteerd in een analytische oplossing voor het in minimale tijdsduur stabiliseren in een nieuwe evenwichtstoestand. De uit deze oplossing volgende regelaar wordt in praktijk al vaak gebruikt. Het regelprobleem om een permanent productietekort (of -overschot) als gevolg van een verandering van evenwichtstoestand te voorkomen, is tevens analytisch opgelost.

Voor de laatste stap van het regelconcept — het koppelen van de continue regelaar aan het fabricagesysteem (dat hier vertegenwoordigd wordt door een DEM) — zijn twee convertors ontworpen: een input convertor, die het continue uitgangssignaal van de regelaar omzet in een geschikt ingangssignaal voor het DEM, en een output convertor, die de uitgang van het DEM omzet in geschikte gegevens voor de regelaar. De input convertor is eenvoudig: de ingaande flux die bepaald wordt door de regelaar, wordt gebruikt als gemiddelde waarde voor de exponentieel verdeelde invoersnelheid van lots. Voor de output convertor is een eerste orde 'non-model based' waarnemer gebruikt, die de ruis (het stochastisch gedrag) uit de output van het DEM filtert en gemiddelde waardes aan de regelaar levert. Met behulp van de convertors is de nl-MPC regelaar succesvol gekoppeld aan het DEM.

Contents

Table of symbols

Abbreviations

CONWIP	CONstant Work-In-Process
DEM	Discrete event model
DES	Discrete event system
ERP	Enterprise resources planning
FIFO	First-in-first-out
JIT	Just-in-time
MPC	Model predictive control
MRP	Material requirements planning
nl-MPC	Nonlinear model predictive control
ODE	Ordinary differential equation
PDE	Partial differential equation
PM	Preventive maintenance
WIP	Work-in-process

Symbols

Δ	Discriminant
δ	Throughput
δ_{ss}	Steady state value of the throughput
λ	Arrival rate
μ	Process rate
ρ	Density
σ	Standard deviation
φ	Flow time
φ_1	Flow time of the first lot
φ_{ss}	Steady state value of the flow time

b	Backlog (negative for surplus, positive for shortage)
c	Coefficient of variation
c_a	Coefficient of variation for inter arrival time
c_p	Coefficient of variation for process time
$cflcond$	Parameter that influences the artificial diffusion
d	Disturbance parameter
e_{rel}	Relative error
k	Sample
m	Number of machines
nx	Number of discretization points in the x-direction
q	Flux
t	Time
t_p	Process time
u	Utilization
v	Velocity
w	Work-in-process
w_{ss}	Steady state value of the work-in-process
x	Place, position in a manufacturing line, degree of completion

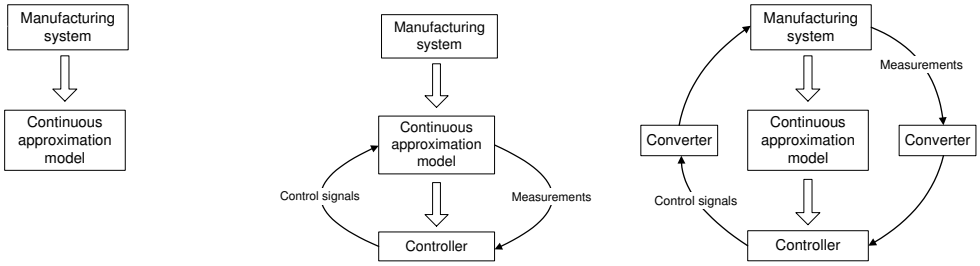
Chapter 1

Introduction

In the field of manufacturing, control is an important issue, which appears at various operation levels. At the tool level, for example, control is necessary in order to assure a properly working tool that processes a product in the desired way. At an intermediate level, sequencing and scheduling rules are used to decide which of the products that are waiting in front of a machine, should be processed first. At the top level of a manufacturing system, the input of the system and the flow of the products through the system are controlled in order to satisfy the customer demands in an 'optimal' way. Here, the definition of 'optimal' is system dependent and can therefore be read in various ways, such as 'with the least possible cost' or 'in the shortest possible time'. In this research, the attention will be focussed only on the top level control as defined above.

In the past, many heuristic methods have been developed for the control of a manufacturing system, such as material requirements planning (MRP), enterprise resources planning (ERP) or just-in-time production (JIT). Nowadays, many heuristic methods are still being used in combination with operator experience for management of resources and planning of production. However, as the complexity of the manufacturing system increases rapidly, the heuristic methods and operator experience will at some point become incapable of finding an optimal control strategy. Furthermore, heuristics can not react on the dynamics of a manufacturing system. Therefore, in this research, a different approach will be studied. This approach is based on (classical and modern) control theory, which is well-developed and widely used for the control of continuous systems.

It is not possible to apply this control theory directly to a manufacturing system. The problem preventing this, is that most control theory is meant for continuous systems, whereas the considered system is not of a continuous nature. In order to be able to apply control theory, the manufacturing system is approximated by a continuous model. Subsequently, a controller is designed for this approximation model. Once a good controller has been developed, it is coupled to the true system. Hereby, the output of the controller should be converted in such as way that it becomes a suitable



(a) Step 1: Continuous approximation model of manufacturing system

(b) Step 2: Controller design for continuous model

(c) Step 3: Controller coupled to manufacturing system using converters

Figure 1.1: The control framework.

input for the manufacturing system, and vice versa. This approach is described in a control framework, which is visualized in Figure 1.1.

An important step in the control framework is to find a suitable approximation model for the manufacturing system. As mentioned before, the model should be continuous. Additionally, it should accurately describe flow time and throughput in both transient and steady state, since these parameters are often key issues in manufacturing control problems. Finally, the model should be solvable within limited computer time.

In literature, roughly three types of models for manufacturing systems can been identified: queueing models (see e.g. [Buz93, Sur98]), discrete event models (see e.g. [Ram87]), and fluid models (see e.g. [Cas02, Har95, Kim83]). Each of these approaches has its own advantages, but none of them is actually suitable for modelling the complete behaviour (transient and steady states) of a manufacturing system. In general, queueing models only deal with steady state behaviour. Discrete event models can be used for the analysis of both transient and steady state behaviour, but they are not of a continuous nature and, especially for large systems, solving the model is time consuming. The fluid models are continuous, but they are throughput oriented and focus mainly on linear time-invariant solutions. Ploegmakers [Plo03] and Rem [Rem03] have investigated the use of fluid models in the presented control framework, considering mainly the throughput. In this research, however, both throughput and flow time will be investigated, and therefore a new approach is needed.

In the theory of traffic simulation and control, partial differential equations (PDEs) are successfully used for modelling dynamic behaviour of highway traffic. Since these traffic models show good correspondence with manufacturing systems (vehicles = products, highway = manufacturing system), partial differential equations may also be useful in the modelling and control of manufacturing systems. The PDE-models are continuous, describe flow time and throughput (flux) in both transient and steady states and they are computationally feasible, which makes them suitable for use in the presented control

framework. However, since the PDE-models have not been introduced into the field of manufacturing until recently, it is still unknown how accurately these models describe parameters such as throughput and flow time. Therefore, the performance of these models should be examined before using them in the control framework.

The second step in the control framework is the design of a controller. This controller, which should be coupled eventually to the manufacturing system, initially has to be designed for the PDE-model that approximates the manufacturing system. In literature, several methods are available to control PDEs. However, not all methods are applicable to each type of PDE or suitable for each type of control problem. In order to obtain a properly working controller it should therefore be investigated which of the available control methods are applicable to the manufacturing PDE-model and suitable for the kind of control problems that are relevant for manufacturing systems.

Finally, in order to complete the control framework, two converters need to be designed. These converters are used to couple the continuous controller to the manufacturing system. A good design of these converters is essential for a good communication between the manufacturing system and the controller, and thus for a good performance of the control framework.

Objectives

This research is part of a larger project, in which the performance of the control framework for manufacturing systems (Figure 1.1) is investigated and compared to the performance of existing control methods. The main objective of this research is to investigate the possibilities of PDE-models in the modelling and control of manufacturing systems. Here, the emphasis is laid on the investigation of the first two steps of the control framework: the modelling of manufacturing systems using PDE-models and the control of these PDE-models. However, in order to complete the control framework, some attention will be paid to the design of the converters as well.

Approach

In order to meet these objectives, the following approach is used, which also defines the outline of this report.

In Chapter 2, first the manufacturing system is considered. Here, main properties are discussed and frequently used parameters are described. Then, in Section 2.2, the case is presented that is used throughout the rest of the report to investigate the possibilities of the PDE-models in simulation and control. The validation model that is used for this investigation is discussed in Section 2.3.

Chapter 3 deals with the basics of partial differential equations and how they can be used in simulation of manufacturing systems. In Section 3.1, a general introduction is

given to PDEs. Then, in Section 3.2, it is explained how the manufacturing system can be seen as a PDE-model and to which demands the PDE-model must comply. Finally, in Section 3.3, three PDE-models are introduced as candidate models for describing the case presented in Chapter 2.

The performance of the candidate models presented in Section 3.3 is evaluated in Chapter 4. For this purpose, first a set-up of experiments is designed, which is described in Section 4.1. The results of these experiments are described in Section 4.2, and evaluated in Section 4.3. In this section, the best candidate model is selected as well. Only this PDE-model is used in the remainder of this research. The chapter is concluded with a discussion, in which the conclusions of this part of the research are placed in a perspective.

The second part of the research, i.e., the search for a suitable controller for the selected PDE-model, starts in Chapter 5. In this chapter, first the control variables are defined in Section 5.1. Then, in Section 5.2, three control problems are defined that are relevant for manufacturing systems. Section 5.3 gives an overview of controllers that can be applied to PDE-models.

The two most promising control methods, Lyapunov's basic stability theory and model predictive control, are evaluated further in Chapter 6. Section 6.1 explains the usage of Lyapunov's basic stability theory and deals with the extra information that is required to solve the defined control problems. At the end of this section, Lyapunov's stability theory is applied to the control problems. In Section 6.2, the aspects of model predictive control are discussed and the control problems are again considered. The chapter is concluded with a discussion, in which the investigated control methods are evaluated and future research on the control of PDE-models is discussed.

In Chapter 7 the control framework (Figure 1.1) is completed and tested. For this purpose, first the design of the two converters is discussed in Section 7.1. In Section 7.2 the control framework is tested on the case described in Section 2.2 and the results are given. In Section 7.3, the test is evaluated and the performance of the control framework is discussed.

Finally, in Chapters 8 and 9, conclusions of this research are drawn and recommendations for future research are made.

Chapter 2

Case: a manufacturing flow line

In this chapter a manufacturing flow line is presented, that is used as a case throughout the rest of this report to investigate the possibilities of PDE-models in both simulation and control. To get a better understanding of what this case stands for, first some general terminology and properties of a manufacturing system are described. In Section 2.2, the case is presented and the corresponding assumptions are given. In Section 2.3, a discrete event model of this case is presented that will be used for the validation of the PDE-models.

2.1 Manufacturing systems

A manufacturing system is an objective-oriented network of processes through which entities flow [Hop00]. The objective of a manufacturing system is to create products out of a selected group of raw materials and semifinished goods. To do so, several processes are present in the system. These processes include physical processes (such as cutting, heating, or assembling), but may also include steps that support the direct manufacturing processes (such as transport, order entry, or maintenance). The entities that flow through the network of processes can be subdivided into two groups: lots and information (that is used to control the system). A lot is a group of products with similar properties and destination in the system. The lots follow a certain routing through the network of processes. This routing is defined as the sequence of processes that the lot should undergo between entering and leaving the system.

Discrete event systems

A manufacturing system can be seen as a discrete event system (DES). Unlike continuous or discrete systems, which are respectively continuous and discrete in time, discrete event systems are based on events. An event is a change in the manufacturing system

that takes place at a certain point in time, such as the arrival of a lot at the buffer in front of a process, or a machine becoming available after maintenance. Note that transport, processing and maintenance themselves are not events, since they do not change the state of the manufacturing system. The start and end of such an action however are events. At each event the state of the manufacturing system changes; between two events the state remains unchanged. Therefore a DES is said to be driven by events.

Common parameters

When describing the properties of a manufacturing system, important parameters are throughput δ, flow time φ and work-in-process (WIP) w. The throughput is defined as the number of lots per unit time that leave the system. The flow time of a (sub)system is the time a lot spends in the (sub)system between entrance and exit. The WIP of a (sub)system describes the amount of lots that is present in the (sub)system at a certain moment. The above three parameters are linked by *Little's Law* [Lit61] for steady state situations:

$$\bar{w} = \bar{\delta} \cdot \bar{\varphi}, \tag{2.1}$$

which states that the mean WIP of a (sub)system equals the product of the mean throughput and the mean flow time of that (sub)system. Other parameters that are also used in the description of a manufacturing system are arrival rate λ (mean number of lots per unit time that enters a certain process or system), process rate μ (mean number of lots that is processed per unit time) and utilization u (degree of machine occupation, i.e., the fraction of the available time that a machine is actually busy processing). For steady state situations, the utilization is defined by:

$$u = \frac{\lambda}{\mu}, \tag{2.2}$$

in which the arrival rate λ should be smaller than (or equal to) the process rate μ (i.e., $u \leq 1$). If the arrival rate is larger than the process rate, the supply of lots is larger than the machine(s) can handle and therefore steady state is never reached.

Variability

Another important concept in describing a manufacturing system is variability. Often process- and transport times are not deterministic, and the cause(s) of the variation in these times can be various: setup time, product differentiation, operator behaviour, maintenance of machines, rework, etc. It is important to describe variability since it has a large impact on the properties of the system, such as flow time and WIP (this will be demonstrated later in this section). The variability of a process is denoted with a dimensionless parameter c: the *coefficient of variation*, which is defined as:

$$c = \frac{\sigma}{\bar{t}_p}. \tag{2.3}$$

Here, $\bar{t}_p$ $(= 1/\mu)$ is the mean process time and σ is the standard deviation on this time. Now using the Pollaczek-Khinchin formula [Khi32, Pol30] from queueing theory, the effect of variability can be (approximately) determined:

$$\bar{\varphi} = \left(\frac{c_a^2 + c_p^2}{2}\right)\left(\frac{u}{1-u}\right)\bar{t}_p + \bar{t}_p. \tag{2.4}$$

Here c_a and c_p are the coefficients of variation for respectively the inter arrival time and the process time. Equation 2.4 is an approximation of the flow time for a G/G/1 system, i.e., a system with a generally distributed inter arrival time (first 'G'), a generally distributed process time (second 'G') and one process. A special case of the G/G/1 system is the M/M/1 system, in which the inter arrival time and process time are exponentially distributed ($c_a = c_p = 1$). For totally deterministic systems ($c_a = c_p = 0$), the first term of the right hand side of (2.4) is zero and thus the mean flow time equals the process time. However, if the system is not deterministic and thus c_a and/or c_p are larger than zero, the mean flow time is larger than the mean process time. This extra time is the mean time a lot has to wait in a buffer in front of the process. The larger c_a and c_p are, the larger this waiting time becomes. Further, it can be seen in (2.4) that for a system with variability, the utilization must be strictly less than 1. As the utilization approaches 1, the mean flow time will go to infinity.

2.2 Case description

A large variety of manufacturing systems exists, ranging from simple flow lines to very complex production systems (such as the semiconductor industry). Since it is impossible to validate the PDE-models for all these kinds of systems, a case is defined for which the PDE-models are evaluated. Of course other cases should be investigated as well before a general conclusion can be drawn concerning the usefulness of PDE-models in simulation and control of manufacturing systems. However, this is left for future research.

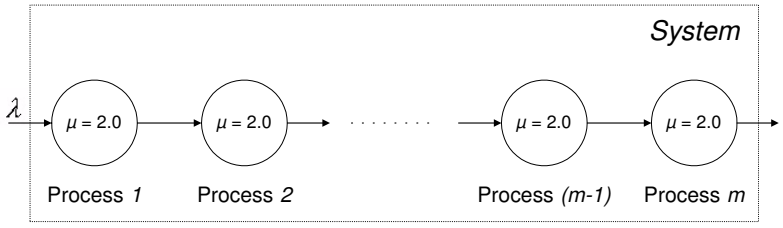


Figure 2.1: Visual representation of the case

The case that will be used for investigation throughout this report is a flow line of identical M/M/1 processes. Each process (workstation) consists of a buffer and a machine. Figure 2.1 gives a visual representation of the case. The case has several specific properties:

- All buffers have infinite capacity.
- All buffers use the first-in-first-out (FIFO) policy.
- Process times are exponentially distributed with a mean of $1/\mu = 0.5$ hours.
- Machines have no setup time and are not subject to failure.
- All machines are single lot machines: only one lot at a time can be processed.
- Processing of lots never fails.
- There is no product differentiation.
- Transport time between processes is negligible.
- The inter arrival time is exponentially distributed, unless stated otherwise.

In order to determine whether the PDE-models can describe this general flow line in a good way, the case is evaluated for two lines with respectively 10 and 50 machines. These lines are subjected to several arrival rates. A detailed description of the instantiations of this case that are investigated for the evaluation of the PDE-models, is given in Section 4.1. The PDE-models have to be validated, in order to determine their performance. The validation model that is used for this purpose is presented in the next section.

2.3 Validation model

The best option to validate models, is of course to use the real-life manufacturing system as validation 'model'. However, since it is time consuming to perform experiments on a real-life manufacturing system, another model that has already proven its good performance, is used instead. As was mentioned in Chapter 1, several models exist for describing a manufacturing system. Of those models, the discrete event model (DEM) describes the dynamics and variability of a manufacturing system accurately. Therefore, a discrete event model is used to represent a manufacturing system and to validate the candidate PDE-models (see Chapters 3 and 4).

In the Systems Engineering Group, the specification language χ [Hof02] is used to create discrete event models. The χ-models consist of concurrent processes (each process describes another part of the manufacturing system) that are linked by channels. The processes are said to be concurrent since they are not evaluated one at a time, but in parallel. The process that generates the next event is evaluated. The state of the system is then changed and the next event is determined. Processes in the model approximate the dynamics and variability of a process in the real-life system by using (samples of) distributions. In Appendix D the DEM (written in χ0.7) can be found

for the case described in Section 2.2. Besides the χ-code, a short explanation of the different processes in the model is given as well.

Now that the test case is defined and the validation model is created, the PDE-models will come to the attention. Chapters 3 and 4 deal with these PDE-models and their performance.

Chapter 3

PDE-models for manufacturing systems

Partial differential equations (PDEs) are a common method to describe the dynamics of a continuum that is dependent on more than one variable. An example of such a continuum is a fluid: its density is a function of 4 variables (3-dimensional space and time). Not only real continua, however, can be described with PDEs. Lighthill and Whitham [Lig55], and Richards [Ric56] brought up the idea to approximate the behaviour of highway traffic with that of a fluid and describe its dynamics with a PDE-model. They interpreted the highway as a 1-dimensional stream (continuum) with cars as particles and ended up with a fairly well PDE-model of highway traffic dynamics. It is only recently that the idea arose to approximate the dynamics of a manufacturing system with a PDE-model [Arm02b, Arm02c, Lef03]. The analogy of the 1-dimensional stream of products through a factory can easily be seen. However, some difficulties should be overcome. In this chapter and in Chapter 4, the suitability of PDE-models for simulation of manufacturing systems is investigated. In Section 3.1, a closer look is taken at PDEs in general. Then, in Section 3.2, the manufacturing system is considered and some basic demands are defined that the PDE-model should satisfy. In Section 3.3, three candidate models are introduced. In Chapter 4, the performance of these candidate PDE-models is evaluated.

3.1 Partial differential equations

Since the features of PDEs are too numerous to mention them all here, only the two most relevant topics for this research are considered, i.e., the classification of PDEs and the way to solve PDEs.

Classification of PDEs

A large variety of PDEs exists, all with different properties and applications. In order to get some structure in this family of PDEs, they can be categorized in three ways: by order, by degree of (non)linearity and by type [McO03, Zac86, Zau89].

In (3.1), a PDE of the i^{th} order is given in its general notation:

$$F\left(x_1,\ldots,x_n,z,\frac{\partial z}{\partial x_1},\ldots,\frac{\partial z}{\partial x_n},\frac{\partial^2 z}{\partial x_1 x_2},\ldots,\frac{\partial^{(i)} z}{\partial x_1^{(i)}},\ldots\right)=0. \tag{3.1}$$

Here $x_1,\ldots,x_n$ are independent variables and z is a dependent variable. The order of a PDE is defined by the highest order partial derivative appearing in the equation. The term(s) with the highest order partial derivative is (are) called the *principle part* of the PDE. Note that other definitions exist for the order and principle part of a PDE. In this report however, the above definitions are used.

For PDEs, the matter of (non)linearity is more complex than it is for ordinary equations. Ordinary equations, such as $y = 2x$ or $\dot{x} = -x^2$, are either linear or nonlinear and a clear distinction can be made between the two. For first order PDEs, Zachmanoglou and Thoe [Zac86] make a distinction between linear, almost linear, quasi-linear and nonlinear. These terms will be explained using a first order PDE in two independent variables x and t and a dependent variable z:

$$F(x,t,z,z_x,z_t)=0. \tag{3.2}$$

Here, z_x and z_t are shorthand notations for respectively $\frac{\partial z}{\partial x}$ and $\frac{\partial z}{\partial t}$. The *linear* form of (3.2) can be written as follows:

$$a(x,t)z_t + b(x,t)z_x + c(x,t)z = d(x,t). \tag{3.3}$$

Here the function F is linear in z_t, z_x and z, with all coefficients (a,b,c and d) depending on the independent variables only. An equation of the form:

$$a(x,t)z_t + b(x,t)z_x = c(x,t,z) \tag{3.4}$$

is called *almost linear* (or *semi-linear*). The function F is again linear in z_t and z_x, with coefficients a and b depending on the independent variables only. However, F is not linear in z. An equation of the form:

$$a(x,t,z)z_t + b(x,t,z)z_x = c(x,t,z) \tag{3.5}$$

is called *quasi-linear*. The function F is still linear in z_t and z_x, but all coefficients are now functions in both the independent variables and the dependent variable. Finally, an equation of the form (3.2) that can not be fit in one of the above classifications is called *nonlinear*.

If two or more first order PDEs are combined, a *system* of first order PDEs results:

$$\mathbf{A}\mathbf{z}_t + \mathbf{B}\mathbf{z}_x + \mathbf{C} = 0, \tag{3.6}$$

in which the matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ contain the coefficients and the vector $\mathbf{z}$ contains the dependent variables. Technically, higher order equations can always be reduced to first order systems, which make systems of first order PDEs stand very closely to higher order PDEs. However, for the sake of simplicity, here this relation between first order systems and higher order PDEs is ignored. In the remainder of this research, only (systems of) first order PDEs will be discussed.

The classification of a system of first order PDEs is similar to the classification of a single first order PDE. For example, if the matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$ are functions of x, t and $\mathbf{z}$, the system is said to be quasi-linear; if the matrices $\mathbf{A}$ and $\mathbf{B}$ are independent of $\mathbf{z}$ and $\mathbf{C}$ is linear in $\mathbf{z}$, then the system is linear.

In order to clarify the hyperbolic, parabolic and elliptic PDE types, a general second order linear PDE in two independent variables is considered:

$$az_{tt} + bz_{tx} + cz_{xx} + dz_t + ez_x + fz + g = 0, \tag{3.7}$$

where a, b, c, d, e, f and g are functions of the variables x and t only. Here it is assumed that at least a, b, c and all their first and second partial derivatives are continuous. The type of a second-order PDE depends on the value of the discriminant Δ:

$$\Delta = b^2 - 4ac. \tag{3.8}$$

Assuming that this discriminant does not change sign in some region R, the PDE is one of the following types in the region R:

$$\Delta > 0: \quad \textit{hyperbolic,} \tag{3.9}$$
$$\Delta = 0: \quad \textit{parabolic,} \tag{3.10}$$
$$\Delta < 0: \quad \textit{elliptic.} \tag{3.11}$$

If the discriminant does change sign in region R, the PDE is said to be of *mixed type* in region R.

For a system of first order PDEs, a similar classification can be made. Recall (3.6) and assume that matrix $\mathbf{A}$ is nonsingular. Multiplying (3.6) by the inverse of $\mathbf{A}$ gives:

$$\mathbf{z}_t + \mathbf{B}'\mathbf{z}_x + \mathbf{C}' = 0. \tag{3.12}$$

The eigenvalues and eigenvectors of matrix $\mathbf{B}'$ now determine the type of (3.12). The type is said to be:

- *hyperbolic*, if all eigenvalues are real, and all corresponding eigenvectors are linearly independent (if all eigenvalues are real and distinct, the eigenvectors are always linearly independent),

- *parabolic*, if all eigenvalues are real, but some eigenvalues are identical and not all eigenvectors are linearly independent,
- *elliptic*, if there are no real eigenvalues.

There are some situations in which none of the above conditions is met (for example: both real and complex eigenvalues). Since these situations fall beyond the scope of this research, they are not discussed here.

A single first order PDE is always hyperbolic [Bro01].

Solving PDEs

When modelling a continuum, it can be desirable to describe the dependent variables as a function of place and time. Most of the time however, this function can not be found explicitly or is hard to derive. A PDE is then used to define the change of these variables in place and time. Together with an initial condition and boundary conditions, the dependent variables can then be determined for each place and at each time.

To solve a first order PDE (or a system of first order PDEs) several methods are available. On the one hand, there are analytical solution methods. These methods, however, are limited in their use, since an explicit analytical solution does not always exist. On the other hand, there are numerical solution methods, which generate a numerical approximation of the solution.

An overview of analytical methods is given in Appendix B. In Appendix B.1, a well-known analytical solution method, called the *method of characteristics*, is elaborated. With the theory of this method, it is derived that under some conditions, the solution of a first order quasi-linear PDE can break due to the 'development' of a discontinuity (see Appendix B.2). This undesirable property should be kept in mind when considering the candidate PDE-models.

In Appendix C, an overview of numerical methods is presented. The numerical method that is called the *fully discrete method* (see Appendix C.1), is used in this research to solve the PDE-models, since it is relatively easy to implement and applicable to various types of PDEs. A disadvantage of this (and other) numerical method(s), however, is that it approximates the solution of the PDEs. The solution determined by the numerical solver is not the true solution of the *considered* PDE, but rather the solution of a PDE that slightly differs from the considered PDE. It is therefore said that the numerical solver adds a small term to the considered PDE, while solving it. In this report, this term is referred to as *artificial diffusion* (see Appendix C.2). Note that because of this artificial diffusion, some PDEs can be numerically solved without problems, whereas the true solution would break down at a discontinuity.

For further reading on the subject of PDEs, the reader is referred to [McO03], [Zac86] and [Zau89].

3.2 Converting the manufacturing system into PDEs

As already mentioned in the introduction of this chapter, PDE-models can be used for the description of more than only real continua. In such a case however, some adjustments have to be made. Recall that, in the example of highway traffic, the highway was assumed to be a 1-dimensional stream and the cars were assumed to be the particles in that stream. For a manufacturing system, such as presented in Chapter 2, similar assumptions can be made. However, these assumptions should not become unreasonable (e.g., in contradiction with the laws of nature). Therefore, it is important to define some specific properties of the manufacturing system that should be observed by the PDE-models.

Assumptions and definitions

If a manufacturing system should be seen as a continuum, this can be done by assuming that the system is a 1-dimensional stream in which the lots are the particles. A problem arises however in defining the velocity and density of this 'continuum'. In contrast to the highway, a manufacturing line is not continuous but rather a chain of machines. It is therefore impossible to define the velocity as the amount of meters per second, or the density as the amount of lots per meter. The root of this problem lies in the definition of place. This will be made clear by an example.

Example 3.1 (The problem of continuity in place) *Assume a manufacturing system consists of 10 identical machines. During the production, a lot visits all machines in line, without backtracking. Assume that the transport time between machines is negligible. The position of the lot in the system is indicated by* x*. At the entrance of the system* $x = 0$ *and at the exit* $x = 1$*. Now, the following problem is considered: if a lot is being processed at machine 6, what is the value of* x*? Two options are examined:*

1. $x = 0.5$ *and it becomes instantaneously* $x = 0.6$ *when it has been finished at machine 6 and arrives at machine 7. This option seems correct, but it implies a discontinuity in place, which is unacceptable, since the PDE is based on continuity in place. This option however can still give a solution under the right conditions: if a system consists of a large number of machines, the step size between two machines becomes small and a continuous solution is approximated.*
2. $x = 0.5$ *when the lot enters the machine and increases during the processing. When the lot has been finished and arrives at machine 7,* x *has reached* 0.6*. Again, this seems correct, but now another problem arises: how does* x *increase during processing? For deterministic process times,* x *can increase proportionally to the process time. For variable process times this can not be done so easily, since the process time is not fixed*

and the variability also causes a waiting time for lots in front of the machine (see Section 2.1). A possible solution to this is to let x increase proportionally to the mean flow time (waiting time + process time) for the lot when it arrives at the machine.

From the above example some conclusions can be drawn. First of all, it is important to have continuity in place. Secondly, when the structure of a manufacturing system is known, fixed points can be set on x that indicate the transition between two subsequent machines. The interval of x between two fixed points describes the time that a lot is present (waiting or being processed) at a machine. For example, in the second option of Example 3.1, the fixed point $x = 0.6$ indicates the transition between machines 6 and 7, while for $0.5 < x < 0.6$ the lot is present at machine 6. Note that it is not important how the relation between the increase of x and the progress of a lot at a machine is defined, as long as there is a clear definition and x is continuous. Based on these conclusions, the definition of place, velocity and density in the PDE-model of a manufacturing system can now be stated:

Definition 3.1 (Place) *The position of a lot in a manufacturing system is indicated with parameter x. At the entrance of the system $x = 0$, at the exit of the system $x = 1$. The entrance and exit of machines in the routing of the lot are indicated with fixed points on x ($0 \leq x \leq 1$). The place between entrance and exit does not only include the process time at the machine, but also waiting time. The exit of a machine therefore is indicated with the same fixed point on x as the entrance of the next machine. Between two fixed points, a lot moves with a certain velocity, that is proportional to the mean flow time for the lot at that machine.*

Note that if the routing of a lot indicates that a machine should be visited more than once, this machine also appears more than once in x. The value of x indicates the position on the routing of the lot and x is therefore sometimes also called the *degree of completion*.

Furthermore, note that the setting of the fixed points on x for the entrance (or exit) of each machine is not unique. Two options are given: the first option is to subdivide x proportional to the number of machines. For example, if there are 10 machines, then the fixed points are $x = 0.0, x = 0.1, \ldots, x = 1.0$. The other option is to subdivide x proportional to the (mean) process time, so that a process with a longer (mean) process time is represented by a longer interval of x. For example, if there are 2 machines with a (mean) process time of respectively 1 and 3 hours, then the fixed point between those machines is set at $x = 0.25$. Note that it is not important which option is used: as long as the setting of the fixed points is known, this can be taken into account when analyzing the results. For the case described in Chapter 2, both options result in the same fixed points, since all machines are identical and all arrivals and process times are exponentially distributed, which causes equal mean waiting times for all machines (see (2.4)).

Definition 3.2 (Velocity) *The velocity of a lot in a manufacturing system is the amount of place that the lot passes within a unit of time.*

Since the increase in place is coupled to the mean flow time, the velocity is always larger than zero: as time passes by, remaining flow time decreases and thus the lot approaches the next fixed point.

Definition 3.3 (Density) *The density in a manufacturing system is the amount of lots per unit of place.*

In obtaining the density in a manufacturing system, the fraction of place in which the density is determined should not be too small. For example, consider a gas continuum: if the fraction of space in which the density is determined, is too small, it contains either a (part of a) molecule or nothing, and the resulting density function is highly discontinuous. If the fraction is larger, the density function is smoother (but still discontinuous, because of the difference in the density values of adjacent fractions) and therefore a better approximation of a continuous function. On the other hand, if the fractions are too large, the density function describes the average density instead of local densities. For a manufacturing system, the same reasoning can be used.

Furthermore, note that the dimensions of a lot are irrelevant in this definition of density. In a gas continuum, the defined fraction can contain a part of a molecule. In the manufacturing system, however, this is not the case, since the unit of place (completion rate) is different from the unit of the dimensions of a lot.

Finally, note that the definitions in this subsection are based on arbitrary choices. In case the place, velocity or density would be defined otherwise, the PDEs describing the system should also be interpreted in a different way.

Physical properties

The PDE-model should accurately represent the manufacturing system. Of course, the ideal situation is that the PDE-model exactly describes the behaviour of the manufacturing system under all circumstances. Although this ideal situation is not strictly required (no model is perfect), some physical properties of the manufacturing system must be accurately described by the PDE-model. These properties are given here:

- Density can not have a negative value. Although this seems trivial, it is stated, since in mathematical models (such as the PDE-model) density is just a variable that can get any value.
- Velocity can not have a negative value, since lots can not travel upstream in the system. For first order PDEs, this is not an issue. For second order PDEs that describe diffusion, however, negative velocities can occur. Daganzo [Dag95]

identified this problem for second order PDE-models of highway traffic. A few years later a solution to this problem was published by amongst others Aw and Rascle [Aw00].

- For flow lines subject to the FIFO policy, the velocity of a lot can not be influenced by other lots that have entered the system at a later point in time. For example, the velocity of a lot that is being processed at the last machine of the line can not be influenced by a pile-up of lots elsewhere upstream in the line.

Requirements for control

Besides the physical demands, the PDE-model must also comply to another important demand: it must be suitable for control purposes. Since the control strategy is not yet defined, it is hard to define specific requirements for the PDE-model. However, some general requirements can be posed:

- The PDE-model must accurately predict both transient and steady state behaviour, for all possible initial states of the system. If the deviation from the manufacturing system is too large, the controller that is based on the PDE-model is not suitable for the real manufacturing system.
- The output of the PDE-model must at least contain information on the throughput and flow time.
- The PDE-model must be solvable within limited time.

3.3 Candidate PDE-models

In literature, three PDE-models have been found that can describe the manufacturing line presented in Chapter 2. In this section, these candidate PDE-models are introduced, described and classified. In Chapter 4, the performance of these models is tested and the (dis)advantages of each model are identified.

Model 1

This model is based on the LWR-model (Lighthill, Whitham and Richards [Lig55, Ric56]), that is used to describe the behaviour of highway traffic. It consists of a mass conservation law and a static relation between velocity and density. In the PDE-model for manufacturing systems [Arm02a], the mass conservation law is defined as:

$$\rho_t + q_x = 0, \tag{3.13}$$

in which ρ and q are the dependent variables, describing respectively density and flux. These variables are dependent on the independent variables t and x, which describe respectively time and place. The flux q can be written as the product of density ρ and velocity v:

$$q = \rho v. \tag{3.14}$$

The static relation between velocity and density has been transformed to a static relation between velocity v and the total WIP w in the system:

$$v(t) = \frac{\mu/m}{1 + \frac{1}{m}w(t)} = \frac{\mu}{m + w(t)}, \tag{3.15}$$

in which μ is the process rate of a machine, m is the number of (identical) machines, and w is defined by:

$$w(t) = \int_0^1 \rho(x,t)dx. \tag{3.16}$$

Equation (3.15) is an 'exact' formula for the mean velocity (inverse of the mean flow time) in steady state, based on queueing theory. The assumption made in LWR-alike models is that this static relation is also valid during the transient state. Equations (3.13) and (3.15) can be linked using (3.14).

With an initial condition $\rho(x, t{=}0)$ and a left boundary condition $q(x{=}0, t)$, this PDE-model has enough information to be solved. It is assumed that everything that reaches the right boundary ($x = 1$) immediately leaves the system (the right boundary is said to be free).

For future reference, the complete model is now restated, with $f(x)$ an initial density distribution and $\lambda(t)$ the arrival rate:

$$\begin{aligned} \rho_t + (\rho v)_x &= 0 \\ v(t) &= \frac{\mu}{m + w(t)} \\ w(t) &= \int_0^1 \rho(x,t)dx \\ \rho(x, t{=}0) &= f(x) \\ q(x{=}0, t) &= \lambda(t) \end{aligned} \tag{3.17}$$

The model described above is a first order (hyperbolic) quasi-linear PDE. Note that the velocity (3.15) depends only on time, not on place. This implies that the velocity in the system is always uniform.

In Section 3.1 it is mentioned that under some conditions, the solution of first order quasi-linear PDEs can break due to the 'development' of a discontinuity in the solution. Because of the uniform velocity, solutions of this quasi-linear PDE can not break (see Appendix B.2). The model can thus be used under all circumstances: both ramp up and ramp down situations can be described.

Model 2

This model consists of two PDEs [Arm02a]: a mass conservation law (3.13) and a similar equation for the development of the velocity:

$$v_t + vv_x = 0, \tag{3.18}$$

which is called the *inviscid Burgers equation*. Besides the initial and boundary conditions described in the previous subsection, here also an initial condition $v(x, t{=}0)$ and a boundary condition $v(x{=}0, t)$ are required for the second PDE (3.18). The right boundary is again assumed to be free. The left boundary condition for (3.18) is computed using (3.15). Note that $v(x{=}0, t)$ is only used as the left boundary condition, and not as the velocity for the whole line (which is the case for Model 1).

For future reference, the complete model is now restated, with $h(x)$ an initial velocity distribution:

$$\begin{aligned} \rho_t + (\rho v)_x &= 0 \\ v_t + vv_x &= 0 \\ \rho(x, t{=}0) &= f(x) \\ v(x, t{=}0) &= h(x) \\ v(x{=}0, t) &= \frac{\mu}{m + w(t)} \\ w(t) &= \int_0^1 \rho(x, t)dx \\ q(x{=}0, t) &= \lambda(t) \end{aligned} \tag{3.19}$$

Note that the first two equations of (3.19) can also be written in matrix notation (3.12):

$$\mathbf{z}_t + \mathbf{B}\mathbf{z}_x = \begin{bmatrix} \rho \\ v \end{bmatrix}_t + \begin{bmatrix} v & \rho \\ 0 & v \end{bmatrix} \begin{bmatrix} \rho \\ v \end{bmatrix}_x = 0. \tag{3.20}$$

The model is a parabolic, quasi-linear system of first order PDEs.

For this model, it is not certain whether the solution can break down (see Appendix B.2). It is observed that, when it is solved using the fully discrete method, no breaking occurs. This, however, can also be the result of the artificial diffusion that is added to the model(see Appendix C.2).

Model 3

This model consists of the mass conservation law (3.13) and a static relation between velocity and density (LWR-alike). The static relation is based on the queueing theory

for an M/M/1 system [Lef03]. Consider an M/M/1 line of m identical machines with process rate μ. The mean flow time $\bar{\varphi}$ of this M/M/1 line in steady state is:

$$\bar{\varphi} = \frac{m}{\mu - \lambda}. \tag{3.21}$$

In steady state, the mean velocity is the inverse of the mean flow time and the flux equals the arrival rate. Therefore, in steady state the relation between velocity and density becomes:

$$\begin{aligned} v(x,t) &= \frac{\mu - q(x,t)}{m} \\ &= \frac{\mu - \rho(x,t)v(x,t)}{m}, \end{aligned}$$

which can be solved for $v(x,t)$:

$$v(x,t) = \frac{\mu/m}{1 + \frac{1}{m}\rho(x,t)} = \frac{\mu}{m + \rho(x,t)}. \tag{3.22}$$

This static relation shows similarities with Equation (3.15). However, the velocity in this equation does not depend on the WIP of the whole system, but on the local density. Therefore, the velocity can, as opposed to the velocity in Model 1, vary through whole the system.

With an initial condition $\rho(x, t{=}0)$ and a left boundary condition $q(x{=}0, t)$, this PDE model has enough information to be solved. The right boundary is again assumed to be free.

For future reference, Model 3 is now completely restated:

$$\begin{aligned} \rho_t + (\rho v)_x &= 0 \\ v(x,t) &= \frac{\mu}{m + \rho(x,t)} \\ \rho(x, t{=}0) &= f(x) \\ q(x{=}0, t) &= \lambda(t) \end{aligned} \tag{3.23}$$

Combining the first two equations of (3.23) gives the PDE:

$$\rho_t + \frac{\mu m}{(m+\rho)^2}\rho_x = 0 \tag{3.24}$$

Model 3 is a first order (hyperbolic) quasi-linear PDE. It is quasi-linear since the coefficient in front of ρ_x in (3.24) is dependent on ρ.

This model can not describe any kind of ramp down situation (see Appendix B.2). Only for steady state or ramp up situations, no discontinuity is 'developed' in the solution. If, however, the model is solved using the fully discrete method, then it can be used for the

description of ramp down situations as well. The artificial diffusion (see Appendix C.2) that is added to the model in this case, prevents the 'development' of discontinuities and hence the solution does not break.

Now that the candidate PDE-models have been introduced, their performance can be investigated. This investigation is described in the next chapter.

Chapter 4

Performance of PDE-models

In this chapter the performance of the candidate PDE-models presented in Section 3.3 is evaluated for the case described in Chapter 2. Section 4.1 presents a set-up of experiments for the investigation of the PDE-models. In this set-up, the instantiations of the case that will be examined are described and the performance measures are defined. In Section 4.2, the results of the experiments are presented. Based on these results the performance of the candidate models is evaluated in Section 4.3. The chapter is concluded with a discussion in which the performance of the candidate PDE-models is placed in a perspective.

4.1 Set-up of experiments

To estimate the performance of the three candidate PDE-models for the case as described in Section 2.2, several instantiations of the case have to be simulated with the PDE-models and validated with the DEM. The PDE-models are solved using the fully discrete method, which is implemented in C^{++} (see Appendix E). The DEM is written in χ and can be found in Appendix D.

The goal of these experiments is to identify good and bad properties of each candidate model, select the best model for now and indicate points of interest that are useful in the search for better models.

The experiments are performed on two systems: the case with 10 identical machines in line and the case with 50 identical machines in line. This is done to investigate whether the number of machines in the flow line influences the accuracy of the solution.

During the experiments, the following questions are examined:

- Does the model go to the true steady state under all circumstances in which a steady state is reached?

- Does the model accurately describe transient behaviour under all circumstances?
- Can the model describe an arbitrary state (or initial condition) well?

These properties are investigated using two kinds of experiments, which are described below. The performance measures that are used for these experiments are discussed at the end of this section.

Before the experiments are specified, first two parameters that are used in the numerical approximation of the PDE-models, are introduced:

- nx: For the numerical approximation with the fully discrete method, the PDE-models are discretized with respect to both place and time (see Appendix C.1). The number of discretization points in x-direction is represented by nx. The step size dx can be derived from this parameter by: $dx = \frac{1}{nx-1}$. The step size in time dt is related to dx, as is described in Appendix C.1.
- *cflcond*: a value within the range $[0, 1]$ that influences the artificial diffusion that is added to the PDE-models by the fully discrete method. For *cflcond* $= 1$, the artificial diffusion is only caused by the effect described in Appendix C.2. For *cflcond* ≤ 1, extra artificial diffusion is added to the approximation model.

Experiment 1: ramp up

In this first experiment, the ramp up of the system is investigated: the empty flow line is injected with an arrival rate that corresponds to a certain machine utilization in steady state. Given a certain target utilization, the corresponding arrival rate can be calculated using (2.2).

The experiment is performed under the following circumstances:

- Four different realizations, with a target utilization of respectively 25%, 50%, 75%, and 95%. This corresponds to an arrival rate of respectively 0.5, 1.0, 1.5, and 1.9 lots per hour.
- The initial state is an empty flow line. For the PDE-models this means that the density is initially zero for all x. For the DEM this means that all buffers are empty and all processes are idle.
- $nx = 101$, this corresponds with a step size dx of 0.01.
- *cflcond* $= 0.5$, unless stated otherwise.

Experiment 2: ramp down

The second experiment deals with the ramp down of the system: the flow line is initially in steady state when instantaneously the arrival rate is decreased to 0.5 lots per hour. Here the WIP in the line is used to describe the initial state: for an M/M/1 flow line in steady state, the mean WIP per process is coupled to the machine utilization by:

$$\bar{w} = \frac{u}{1-u}. \tag{4.1}$$

For example, for a system in steady state with a machine utilization of 50%, the initial WIP is exactly one lot per process. For the DEM this means that the initial content of all buffers is set to one. For the PDE-model, the initial density is derived using the following reasoning: if there is one lot per process in steady state, and there are m processes in the flow line, then altogether there are m lots in the line. Recalling that the flow line is described at the interval $[0, 1]$, the density in steady state must be m for all x, so that (3.16) is satisfied.

The experiment is performed under the following circumstances:

- Four different realizations with an initial steady state that is characterized by a machine utilization of respectively 50%, 75%, 90%, and 95%. This corresponds to an initial WIP of respectively 1, 3, 9 and 19 lots per buffer for the DEM and to an initial density of respectively m, $3m$, $9m$ and $19m$ for all x in case of the PDE-models.
- The target utilization is 25%, this corresponds to an arrival rate of 0.5 lots per hour.
- $nx = 501$, this corresponds with a step size dx of 0.002.
- $cflcond = 0.5$, unless stated otherwise.

Note that for experiment 2, nx has been chosen larger than for experiment 1. During the experiments, it has been observed that, especially for low utilizations, the solution gets more accurate by increasing nx. Since experiment 2 is, in comparison with experiment 1, more concerned with low utilizations (all solutions go to an utilization of 25%), nx is chosen larger.

Performance measures

In the described experiments, the following parameters are measured:

- The value of the mean WIP, the mean throughput, and the mean flow time once the (new) steady state has been reached (respectively w_{ss}, δ_{ss} and φ_{ss}).

- The time that has passed before w_{ss}, δ_{ss} and φ_{ss} are reached. Here it is assumed that steady state is reached once the variable remains within 99% and 101% of the steady state value.

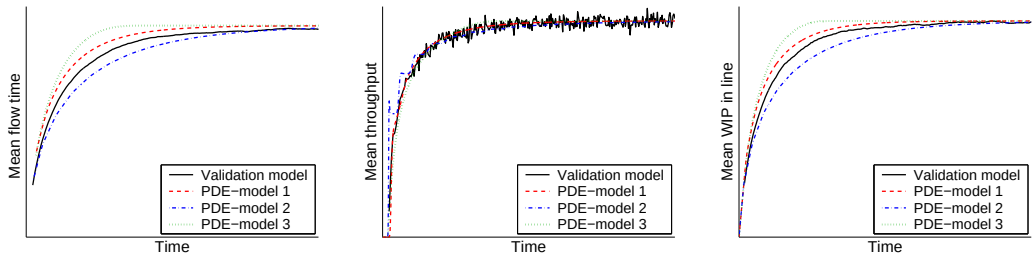
- The flow time value of the first lot φ_1. Here, the first lot is defined as the first new lot that enters the system for $t \geq 0$.

- The computational effort that is required to solve the model.

In order to validate the PDE-models, the simulation results of these models should be compared to those of the validation model. However, the models generate a different kind of output data: the output data of the PDE-models contains data on the density, velocity and flux that can only be translated into *mean values* of WIP, throughput and flow time. The output data of the DEM, on the other hand, contains highly variable data on WIP, throughput and flow time. In order to be able to compare the results of the PDE-models with those of the DEM, the output of both the PDE-models and the DEM is processed. The methods that are used for this output processing are described in Appendix F. Once the output is processed, the behaviour of the PDE-models can be compared with that of the DEM, using the performance measures as defined above. The results of this performance study are presented in the following section.

4.2 Results

In this section a summary is presented of the results obtained from the experiments as described in Section 4.1. The complete set of results can be found in Appendix G.



(a) Mean flow time (b) Mean throughput (c) Mean WIP in line

Figure 4.1: Typical results for the ramp up experiment

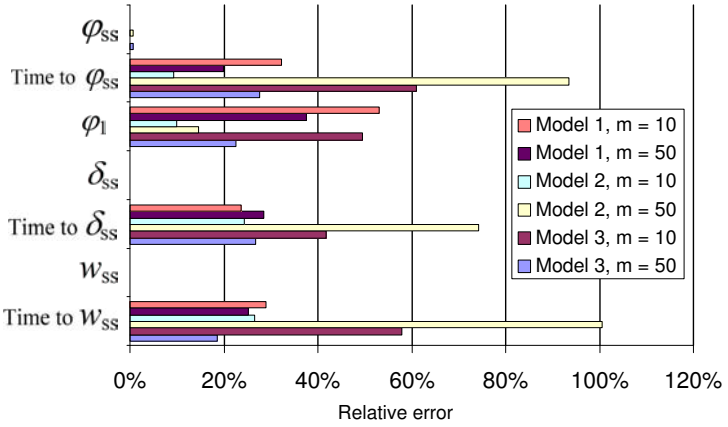


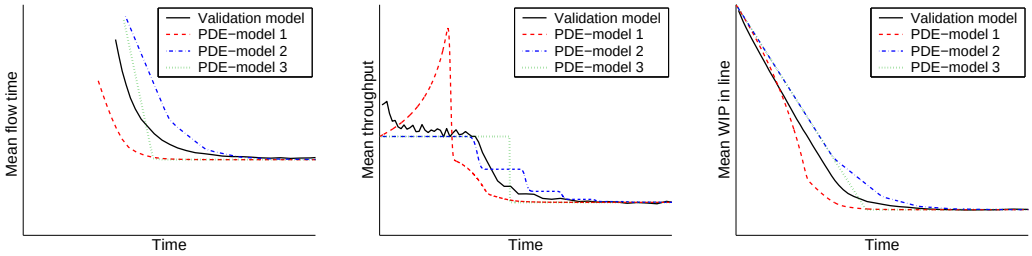
Figure 4.2: Performance measures: the average result for the ramp up experiment

First, the results of the ramp up experiment are considered. A typical result is visualized in Figure 4.1. Here, the development in time of the mean flow time, mean throughput and mean WIP in the line is plotted for the three candidate PDE-models and the validation model. It can be seen that the steady state is accurately described by the PDE-models. In the transient state, however, the results of the PDE-models significantly differ from those of the DEM. This observation can also be obtained from Figure 4.2, in which for each performance measure (Section 4.1) is depicted how large the deviation of the PDE-models is from the DEM, when averaged over the four instantiations of the experiment. Here, (4.2) was used to calculate the relative error e_{rel} between the PDE-model and the DEM:

$$e_{\text{rel}} = \frac{\|value_{\text{DEM}} - value_{\text{PDE}}\|}{value_{\text{DEM}}} \cdot 100\%. \tag{4.2}$$

For the second experiment, i.e., the ramp down experiment, typical results are plotted in Figure 4.3. Note that the curves of the mean flow time (Figure 4.3(a)) only appear after a while, since the first flow time that can be determined is the one from the first released lot after $t = 0$. From Figure 4.3 it can again be concluded that the steady state is accurately described by the PDE-models, while the description of the transient state shows some significant errors. These errors are also indicated in Figure 4.4, in which for each performance measure (Section 4.1) the average deviation of the PDE-models from the DEM is visualized.

Note that the performance measure 'computational effort' is not included in Figures 4.2 and 4.4, since the computer times required to solve the PDE-models can not be compared to those required for the DEM (the DEM requires several sets of simulations to obtain a 'mean simulation result'). The computer times for the PDE-models, nevertheless, are included in Appendix G.



(a) Mean flow time (b) Mean throughput (c) Mean WIP in line

Figure 4.3: Typical results for the ramp down experiment

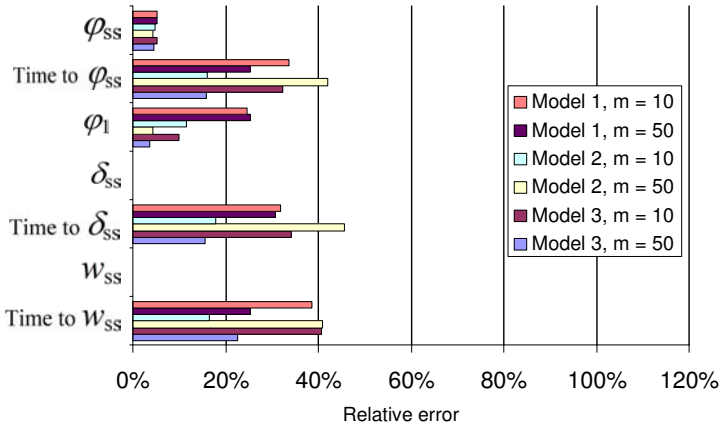


Figure 4.4: Performance measures: the average result for the ramp down experiment

Besides the observations that were already mentioned in this section, several other observations have been made during the experiments. These observations include general evaluation issues as well as peculiarities that need further investigation. In the following section, these observation are discussed during the evaluation of the three candidate PDE-models.

4.3 Evaluation of the PDE-models

As stated in Section 4.1, the goal of the performed experiments was to gain more information in the strong and weak points of the PDE-models, to point out interesting fields for future research and to select one of the candidate PDE-models for use in the remainder of this research project. In order to meet this goal, the candidate PDE-models are evaluated one by one. For each model, the information that was derived from the experiments is discussed and strong and weak points are determined. Besides

the results of the experiments, some general remarks are taken into account as well. At the end of this section, the pros and cons of each model are weighed and compared to those of the other models. Based on this comparison, one of the candidate models is selected for use in the remaining part of this investigation, i.e., the control of the flow line.

Model 1

From the results of the experiments it can be concluded that Model 1 (3.17) stabilizes at the correct steady state value, under all considered circumstances. The transient state, however, differs much from the one described by the validation model. In most considered instantiations of the experiments, this PDE-model describes the transient too fast: the time to reach stability is too short. In some instantiations (for example: ramp up, $m = 50$), however, the transient is too long. Further investigation and more experiments are required to find a valid explanation for this observation.

A remarkable property of this model is that it has a uniform velocity throughout the system at all times. On the one hand, this has its advantages. All initial states can be described easily once the initial density is known, since the initial velocity can be determined from the total WIP in the system. Furthermore, no breaking of the solution can occur (see Appendix B.2), since the velocity is uniform at all times, and thus both ramp up and ramp down can be described under all circumstances. On the other hand, it also has its disadvantages. The main disadvantage can be seen in Figure 4.3(b): for ramp down situations, the throughput first increases before it decreases to the new steady state value. This effect is caused by the model property that the velocity uniformly increases for decreasing WIP. In case of a ramp down, the density in the front-end of the line directly decreases, while the density in the back-end of the line only decreases after a while. So, while the velocity in the whole line increases, the density in the back-end of the line remains constant for a period. By the definition of (3.14), the throughput (outflux) is the product of the velocity and the density at the end of the line, which explains the temporary increase in Figure 4.3(b). Note that this phenomena also occurs during ramp up: the throughput first decreases before it increases to the new steady state value. In the considered ramp up experiments, however, this result is not found, since the density initially is zero. From this observation, it can be concluded that the uniform velocity feature is a bad property, since the model fails to meet an important demand (see Section 3.2): lots are influenced by other lots that enter the system at a later point in time, whereas they should not be.

Furthermore, the mean flow time of the first lot φ_1 is also influenced by this uniform velocity. For ramp up, φ_1 should equal the inverse of the maximum velocity. Initially, the first lot has maximum velocity, however once more lots have entered the system, the velocity drops for all lots and thus φ_1 becomes too large. For ramp down, the same reasoning can be used: since the total WIP decreases, the velocity of all lots in the system increases, and thus φ_1 gets too small. Although this uniform velocity property

has a large influence, it is concluded from the simulation results (Appendix G) that it is not the only factor that affects φ_1. It is left for future research to identify the other factors and their influence.

Finally, an advantage of this model is that it is relatively simple, for it consists only of one PDE and a static relation. On the other hand, this relation between the velocity and the density of the system is nonlinear, which makes the PDE-model more difficult to solve. Furthermore, because of this simplicity and especially because of the uniform velocity property, it is harder to investigate more complex systems, such as a line with heterogene processes or non-exponential distributions.

Model 2

This model (3.19) shows an accurate description of the steady state behaviour and a good stabilization at the desired steady state value. In the description of the transient state, however, the model shows large deviations from the description by the DEM. In general, the duration of the transient state is too long. For some instantiations of the experiments (for example: ramp down, $m = 10$), however, this general observation does not hold. Furthermore, it is observed that in general φ_1 is overestimated, while for the ramp up experiment with $m = 50$ it is underestimated. To explain these observations, further research is required on the description of the transient.

The property that distinguishes this model from the other candidate models is the second PDE, which describes the development of the velocity independently of the density. It is only related to the density by the (left-)boundary condition. An advantage of this second PDE is that the velocity can vary throughout the system. However, it remains to be investigated whether this property can lead to a discontinuity in the solution (see Appendix B.2). During the experiments, no discontinuity occurred, but possible discontinuities might have been undone by the artificial diffusion that is added to the model by using the fully discrete method, which is used to solve the PDEs. A disadvantage of this second PDE is that the velocity is related to the density *only* through the boundary condition. If, for example, the first workstations in the line are empty, while the rest of line is full of lots, a new lot that enters the line initially obtains a velocity that is lower than the maximum. However, the emptiness of the first workstations should result in a maximal initial velocity for the new lot. Another disadvantage of this second PDE is that it is harder to describe an initial state, because the velocity is (almost) independent of the density. Although the initial density function can be derived easily from the WIP data in the system for an arbitrary initial state, the initial velocity function is not obtained so easily, since velocity data can not be measured instantaneously. The system should be observed for some time before an initial velocity function can be obtained. From these observations, it can be concluded that the separate PDE in Model 2 for the description of the velocity causes the model to be unsuitable for the description of certain situations.

Furthermore, it can be seen in Figures 4.1(b) and 4.3(b) that the throughput described by Model 2 shows a 'staircase' behaviour. It is still uncertain how this behaviour is caused, although it is assumed that it is related to the discontinuity in the influx (from 0 to λ at $t = 0$). Further investigation is required to validate this assumption.

Finally, an advantage of this model is that it can be extended to describe more complex systems. One way to extend the model is to add terms to the right hand side of the velocity-PDE, in order to make the velocity dependent on the density or to describe for example CONWIP (CONstant WIP) behaviour. Another possible extension is to add a third PDE to the model in order to describe variability, as described in for example [Hoo00]. These extensions need to be examined further. A final disadvantage of this model is that it consists of a *system* of PDEs, which makes it more difficult to solve then a single PDE, such as in the other candidate models.

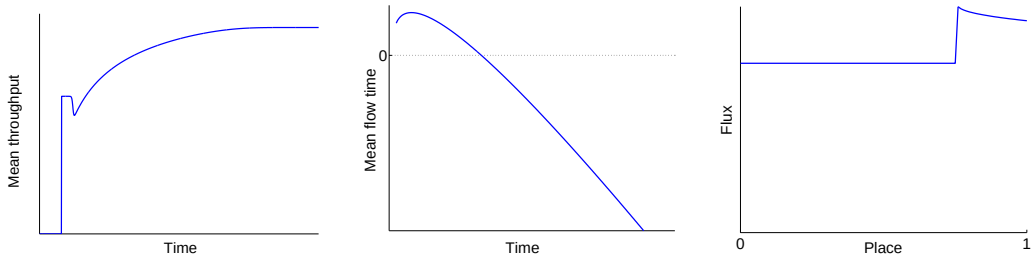
Model 3

Model 3 (3.23) shows an accurate steady state description and a transient state that leaves much to be desired. For this model it can be seen that the transient is too fast under all considered circumstances, without exceptions: the time to reach steady state is shorter than the time predicted by the validation model. The flow time of the first lot, however, is overestimated for all considered instantiations. Further research is needed to investigate these deviations.

The property that distinguishes this model from the other candidate PDE-models is that it describes the velocity as a function of the local density (3.22), which implies that the velocity is a function of both time and place. This is a good property, since in reality the velocity also changes in time and place. An other advantage of this model is that it allows an easy description of an arbitrary initial state: only an initial density function is required, which can be derived easily from the WIP in the system. A disadvantage of this model is that because of the nonuniform velocity a discontinuity is 'developed' in the solution for ramp down situations (see Appendix B.2). During the experiments, however, the ramp down situations could be simulated with Model 3 without discontinuities, because of the artificial diffusion that is added to this model by the fully discrete method, which is used to solve the PDE.

Strangely enough, during *ramp up* a singularity does occur for the instantiations with $u = 75\%$ and $u = 95\%$. After examination, it appeared that this singularity is related to the value of *cflcond*. For the mentioned instantiations, there exists a 'border region' for the value of *cflcond* above which a singularity occurs in the throughput (Figure 4.5(a)) and above which the accuracy of φ_1 is very dependent on *cflcond*. The other performance parameters are hardly affected by the value of *cflcond*. Below this border region, φ_1 is more accurate and almost independent of *cflcond*. In the border region itself, a singularity occurs which causes a negative flow time (Figure 4.5(b)) and a throughput that is higher than the influx (Figure 4.5(c)). It remains to be investigated why this sin-

gularity occurs and why it only occurs for ramp up situations. The results as presented in Section 4.2 were obtained using a value of *cflcond* below the border region.



(a) Throughput, *cflcond* above 'border region'

(b) Flow time, *cflcond* in 'border region'

(c) Flux, *cflcond* in 'border region'

Figure 4.5: Singularities during ramp up simulations with Model 3

During the experiments with Model 3 it also has been observed that the development of the density in the system can be characterized as a steep front moving from the entrance to the exit of the line. Since the density is directly related to the throughput (outflux) by (3.14) and (3.22), this steep front also appears in the throughput of the system (Figures 4.1(b) and 4.3(b)). It is assumed that this steep front is caused by a lack of diffusion in the model. If somehow the model could describe more diffusion, then the front would be less steep and the solution would thus be closer to that of the validation model. Further research is however required to investigate whether this assumption is valid and how extra diffusion can be added to the model.

Furthermore, an advantage of this model is that it is relatively simple: it consists only of one PDE and a static relation. Another advantage is that the model's process rate can be made dependent on place and time, which is useful for the description of a line with different kinds of processes. A final advantage of Model 3 is that it can be adapted in order to describe more complex systems than the considered M/M/1 line. For example, in stead of using the M/M/1 relation from queueing theory to obtain the relation between velocity and density (3.22), the G/G/1 or G/G/m relation from queueing theory might be used. These options, however, have not yet been examined thoroughly.

Selection of the PDE-model

In order to select a model for use in simulation and control of a manufacturing system, the candidate models are compared, based on the strong and weak points of each model.

Model 1 is disregarded, since it is not able to compete with the performance of Models 2 and 3, because of its velocity that is uniform in x. The model can be used for rough estimations, but is not suitable for detailed investigations or complex systems.

The performance of Models 2 and 3 is comparable. Besides the performance, both models have also strong and weak points that are of comparable importance. However, since Model 3 is less complex than Model 2, it will be used in the remaining investigations.

4.4 Discussion

In this chapter, three PDE-models have been evaluated on their performance in simulation of manufacturing systems. The main results of this evaluation can be summarized as follows: during ramp up and ramp down experiments, the models converge to the correct steady state. Approximation of the transient state, however, is highly inaccurate. The evaluation further shows that the considered models all have their peculiarities, which cause some behaviour that is unnatural for a manufacturing system. For each model, this is illustrated by an example. In Model 1 it is assumed that the velocity in the system is uniform in x under all conditions, which causes the unnatural behaviour that the velocity of a lot is dependent on lots behind it. For Model 2, the boundary condition for the velocity is dependent on the total WIP in the system. In the situation that the whole system is empty except for a large amount of lots at the last process, the starting velocity of a new entering lot would be far below the maximum, while the processes in front of it are empty. Model 3 can not describe a ramp down situation. However, the numerical approximation of this model, in which artificial diffusion is added to the model, can describe the ramp down situations.

Model 3 has been selected for use in the investigation of PDE-models in control, since it was the most suitable model out of the three available models. However, the question can be posed: is this model good enough for usage in the control framework? Does this model comply to the general demands, or is it just the best choice out of three bad models? As mentioned before, the description of the transient state leaves much to be desired. On the other hand, a transient fase is described, which is a step in the good direction, since other approximation models, such as flow models and queueing models, in general do not describe this at all. Therefore, for now this model will be used in the investigation of PDE-models in control. Nevertheless, the search for a better model should go on, with the focus on the description of transient behaviour. Hereby, possible causes of the deviations in transient state should be scrutinized.

Another point of attention for future research is the method to solve the PDEs. During this investigation, the fully discrete method has been used, which adds (under some conditions) artificial diffusion to the models. Because of this artificial diffusion, the numerical approximation of the models describes other behaviour than the PDE-model itself would do. An obvious example of this difference can be seen for Model 3: the model itself is unable to describe a ramp down, whereas the numerical approximation has no problem with it at all. This difference, which is also present for Model 2 (but not for Model 1, see Appendix C.2), also causes trouble in the control of the PDEs, since a controller is designed for the PDE-model, but evaluated with the approximation

of the model. Despite of this, for now the fully discrete approximation method is used during the design of the controller. Hereby, it is kept in mind that this difference might be a cause for particularities in the performance of the controller. For future research, however, it is recommended to investigate whether this problem can be avoided by the use of another PDE-solver.

Finally, an interesting question that remains is the applicability of the current PDE-models. Does it accurately describe the behaviour of lots at one machine, 10 machines or 1000 machines? Does the utilization have influence on the accuracy of the solution? Are there other parameters that influence the performance of the PDE, or is the PDE suitable and accurate under all circumstances? From the results of the experiments, it has already become clear that the number of machines, the utilization and nx have impact on the performance on the models. However, in order to describe trends in this influence, the experiments need to be extended. This can be done for example by repeating the current simulations with respectively different numbers of machines in the line, different levels of utilization, and different values for nx. The information on trends in the simulation and dependence on parameters, can be very useful for a better understanding of the PDE-models and is therefore essential in the development of better PDE-models.

In the remainder of this research, Model 3 as well as the fully discrete approximation method will be used, consciously of the discussion points presented in this section.

Chapter 5

Control of the PDE-model

In this chapter and Chapter 6, the controllability of the selected PDE-model (3.23) is evaluated. First the relevant input and output variables of the PDE-model are defined in Section 5.1. Then, in Section 5.2, three control problems are presented that are relevant for manufacturing lines. The chapter is concluded with a description of control methods that can be applied to a PDE-model. The two most promising control methods are discussed further in Chapter 6.

5.1 Controller variables

In order to design a good controller, it is important to know which input variables are available and which of those can be used to influence the state of the model. It is equally important to know which output variables are available and which of those can be used to determine the state of the model. For the case (see Chapter 2) that is described by the selected PDE-model, the following in- and output variables are available:

Input variables	Output variables
• Influx	• Flux
• Process rate	• Velocity
	• Density

Although these variables (except the influx) can be measured or altered at each infinitesimally small fraction of x, it must be kept in mind that this can not be done for the real-life manufacturing line. Therefore, the controller variables should be chosen in such a way that once a controller has been designed, it can be coupled to the real-life manufacturing system without fundamentally changing these variables. In the following subsections, the in- and output variables are investigated further and a set of suitable controller variables is selected.

Input variables

The influx (arrival rate) is a common control variable, which can be used in both the PDE-model and the real-life manufacturing line. Moreover, lots that enter the line travel through the whole system and therefore influence all parts of the state. Since the influx controls the entrance of lots in the line, it can influence the state. It is therefore a suitable input variable.

In reality, the process rate of a machine can not be changed, since it is a property of the machine. However, the process rate can be decreased artificially by leaving the machine idle for some time, while a lot is waiting in front of it. Another artificial way to decrease the process rate is by introducing a maximum number of lots that is allowed to be processed per period of time. Of course, this number should be restrictive in order to cause a process rate decrease. In the PDE-model, it is not possible to leave a machine idle or to introduce a maximum on the number of produced lots per period of time. However, here the process rate can be altered, since it is just a modelling parameter. By modifying the process rate in a part of the manufacturing line, the outflux (throughput) of that part changes, which effects the density (WIP) in that part and all parts downstream in the line. For this reason, the process rate can be a useful input variable, although the difference between its use in the PDE-model and the real-life manufacturing line might cause some difficulties.

In order to keep the controller simple, it has been decided to use only the influx as input variable, and leave the process rate at a constant value.

Output variables

Before selecting the suitable output variable(s) that can determine the state of the system, it is important to define the state of a system.

Definition 5.1 (State) *The state of a system is a set of current (and past) system data that, in combination with all current and future inputs, allows computation of an arbitrary future output of the system.*

At first sight, one could think that the flux data of the whole system is not enough data to describe the entire state, since only the change of density can be determined with these data, but not the absolute value of the density. For the selected PDE-model, however, the flux as well as the velocity are functions of the density. If one of the three mentioned variables is known, the other two can be directly derived from it, using (3.14) and (3.22). Therefore, the state can be described by either the flux, the velocity or the density in the system. However, since it is hard to determine the velocity or the flux at each moment in the real-life manufacturing line, the density is selected as output variable. The only problem that remains is that conversion of the WIP, which can be measured at all times in the real-life manufacturing, into density might give some

difficulties. This problem is discussed further in Chapter 7, in which the coupling of the controller to the discrete event model is described.

5.2 Control problems

For the design of a controller, three control problems have been considered, that are relevant for manufacturing lines. The problems, which are ranged in order of increasing complexity, are described below. In Chapter 6 a controller is derived for these control problems using two different control methods.

Problem 1: Convergence to steady state

The simplest control problem related to the production planning of a manufacturing line, is to bring the system from one steady state, which corresponds to a certain production rate, to another steady state [Arm02c]. An example of a situation in which this control problem appears, is the ramp up or ramp down of a manufacturing line that is performed to meet a new constant customer demand. Assuming the customer demand d changes in the following way:

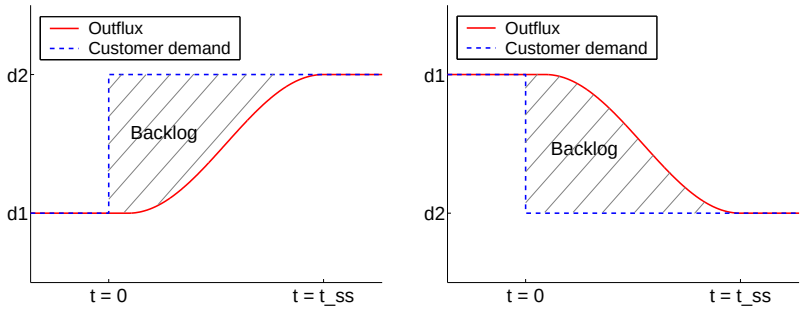
$$d(t) = \begin{cases} d_1 & \text{if } t < 0 \\ d_2 & \text{if } t \geq 0 \end{cases}, \tag{5.1}$$

where $d(t) < \mu$ for all values of t, then the time optimal control problem can be stated as follows:

> *Design the influx as a function of time, such that the system moves from one steady state corresponding to the outflux d_1, to another steady state corresponding to the outflux d_2 in the shortest possible time.*

Problem 2: Convergence to steady state without backlog

When a successful controller has been derived for control problem 1, the resulting outflux as a function of time will probably take a form similar to the one depicted in Figure 5.1(a) for ramp up or Figure 5.1(b) for ramp down. Here, the red line represents the actual system outflux, whereas the blue line denotes the customer demand. It can be seen that between $t = 0$ and $t = t_{ss}$ the outflux is not equal to the customer demand, which causes a positive backlog (shortage of products) for ramp up situations or a negative backlog (surplus of products) for ramp down situations. Once the new steady state is reached, the outflux equals to the customer demand per unit of time, so that the backlog remains.



(a) Ramp up (b) Ramp down

Figure 5.1: Backlog caused by moving the system to another steady state

The described backlog is undesirable and therefore it will be taken into account in control problem 2 as an extra demand: once the new steady state has been reached, the backlog must be zero. The full description of control problem 2 thus reads:

> *Design the influx as a function of time, such that the system moves from one steady state corresponding to the outflux d_1, to another steady state corresponding to the outflux d_2 in the shortest possible time, without creating a permanent backlog.*

Problem 3: Convergence to steady state without backlog, with PM

Control problems 1 and 2 are only concerned with idealistic machines that never fail and have no need for maintenance. Control problem 3 studies the implication of maintenance (for example: cleaning, lubrication, calibration). During preventive maintenance (PM), the machine can not produce lots. Consider the manufacturing line with 10 machines as introduced in Section 2.2, but now with machine 5 subject to PM: each 10^{th} hour the production of this machine is stopped for a duration of 1 hour. Here, it is assumed that the production of a lot can be interrupted by PM and resumed after the PM without any consequences. Is it, under these circumstances, (still) possible to solve control problem 2? In order to find a solution to that question, control problem 3 is defined as:

> *Design the influx as a function of time, such that the adapted system (with machine 5 subject to 1 hour of PM every 10^{th} hour) moves from one steady state corresponding to the outflux d_1, to another steady state corresponding to the outflux d_2 in the shortest possible time, without creating a permanent backlog.*

5.3 Control methods overview

In the past, several methods have been developed for the control of systems described by partial differential equations. However, not all control methods are applicable to each type of PDE or to each type of control problem. Most control methods discussed in literature are focussed on the feedback control of PDEs that are internally controllable (i.e., with a control variable in the PDE itself) [Chr96, Chr98, Gun98, Sir89, Wan66], whereas other methods deal with the boundary control of a PDE [Baş98, Cor02, Sch01]. Since it has been decided in Section 5.1 to use only the influx (i.e., the left boundary condition) as a control variable, control methods for the stated control problems (Section 5.2) should be based on boundary control.

In the remainder of this section, several control methods that might be applicable to boundary control problems, are presented and their suitability to control the selected PDE-model is discussed. A summary of these control methods is given in Table 5.1: the first column gives the name of the control method, the second and third column indicate whether the PDE-model needs to be linearized respectively discretized before the control method can be applied, and the fourth column gives a specific property of the control method.

Control method	**Linearization required**	**Discretization required**	**Characteristic property**
Lyapunov	no	no	Output is stabilized at desired value by means of a Lyapunov function
MPC	no	yes	Internal model predicts system output for various inputs
Feedback linearization	no	yes	Nonlinear feedback yields linear input-output relation
Inverse system	no	no	Inverse PDE switches system's in- and output
Transfer function	yes	no	Input-output relation in frequency domain

Table 5.1: Summary of boundary control methods

Lyapunov's basic stability theory

Lyapunov has made a large contribution to the research on stability of dynamical systems with his work [Lya92]. His basic stability theory is based on the so-called 'least total energy'-principle, which says that a system of bodies is at a stable equilibrium if it is in a point of (locally) minimal total energy. Lyapunov gave a precise characterization

of functions that qualify as 'valid energy functions' and the notion that a dynamical system, whose 'energy' is represented by such a Lyapunov function, stabilizes if this function decreases along the trajectory of the system [Sas99].

Since 'valid energy functions' can be found for the selected PDE-model, it might be possible to use Lyapunov's theory to stabilize the PDE-model at a certain (new) steady state. Note that with Lyapunov's theory, a system can be *stabilized*, not controlled. However, since the considered control problems (Section 5.2) are mainly concerned with the stabilization of the system at a new steady state, Lyapunov's theory might be useful. It will therefore be considered further in Chapter 6.

Model predictive control (MPC)

Model predictive control is a relatively young control method which uses a dynamical model of the process to be controlled [Ess02b]. With this internal model, predictions are generated for the future behaviour of the process. Based on this prediction, a goal function (which usually contains terms as setpoint-deviation and input effort) is optimized with regard to the current and future control inputs. Since a deviation between the model prediction and the real process output may occur (perfect models do not exist), the model is updated and the optimization is repeated at each time sample. In this way, MPC can be seen as an optimal controller with respect to the chosen goal function.

MPC can be used on the selected PDE-model, provided that at least for the internal model, the PDE-model is discretized in place and time. In Chapter 6, MPC is used to search for a solution to the control problems stated in Section 5.2.

Feedback linearization

Nonlinear systems are in general harder to control than linear systems, because of their complex input-output relation. For some nonlinear systems however, a (local) linear input-output relation can be established by adding a nonlinear feedback term to the input signal [Sas99]. This is clarified with the following example:

Example 5.1 (Feedback linearization) *Consider the following nonlinear system with state variable x, input u and output y:*

$$\begin{aligned} \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} &= \begin{bmatrix} \sin(x_1) + x_1^3 + x_2 \\ u \end{bmatrix}, \\ y &= x_1. \end{aligned}$$

First a state-coordinate change is performed with:

$$\begin{aligned} z_1 &= y = x_1 \\ z_2 &= \dot{y} = \dot{x}_1 = \sin(x_1) + x_1^3 + x_2 \end{aligned}$$

so that:

$$\begin{bmatrix}\dot{z}_1\\ \dot{z}_2\end{bmatrix} = \begin{bmatrix}\dot{x}_1\\ \big(\cos(x_1)+3x_1^2\big)\dot{x}_1+\dot{x}_2\end{bmatrix} = \begin{bmatrix}z_2\\ \big(\cos(z_1)+3z_1^2\big)z_2+u\end{bmatrix}.$$

Now, by choosing the input u as a function of the new coordinates z and a new input u^*, an input-output linearizing feedback is obtained:

$$u = -\big(\cos(z_1)+3z_1^2\big)z_2 + u^*,$$

which gives the following linear input-output relation:

$$\ddot{y} = u^*$$

The selected PDE-model also can be linearized by feedback, provided that it is discretized in place and time. This proof is given in Appendix A.1, based on the conditions as stated in [Lee87]. For the considered PDE-model, feedback linearization is however not advantageous in control, since the constraints of the original PDE-model ($\lambda \geq 0$, $\rho \geq 0$) can not be translated into simple constraints for the linearized system. Consider for example the feedback signal in Example 5.1. If input u is simply constrained by an upper and lower bound, then in order to comply with these constraints, u^* must be constrained as well. The resulting constraints on u^*, however, are nonlinear and dependent on z_1 and z_2, which makes it difficult to find a good controller. Regarding the selected PDE-model, the constraints for the linearized system are even more complex which causes the advantage of a linear input-output relation to be cancelled out completely. Feedback linearization is therefore not considered further in this research.

Inverse system

The concept of an inverse system is that it uses the known required output (customer demand) of the original system, as input in order to determine an output, which is in its turn a possible input for the original system. This concept is visualized in Figure 5.2. The problem, however, with inverse systems is that they might come up with an input for the original system that is not robust (a small deviation in the input might cause a totally different output), or that they might come up with multiple inputs. Furthermore, an inverse system does not necessarily exist for all systems. In order to decide whether this method is suitable for the control of the PDE-model, it first has to be found out under which circumstances the inverse system exist and can provide a unique and robust input. However, this is left for future research. For now, this option is not considered further.

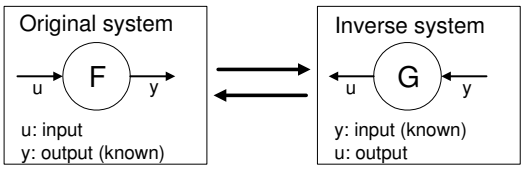


Figure 5.2: Concept behind inverse systems

Transfer function

A transfer function is a direct relation between in- and output in the frequency domain. In order to determine a transfer function for a PDE-system [Rab99], the system should be linear. Linearization of the selected PDE-model is however not an option, since the PDE-model is highly nonlinear and linearization would harm the description of the transient state. Therefore, this method is not considered further.

Other boundary control methods

Besides the already mentioned control methods, two different control methods for PDEs can be found in literature: Hamiltonian boundary control systems [Sch01] and H^{∞}-Optimal Boundary Control [Baş98]. These methods have not been investigated yet and are left for future research.

Chapter 6

Performance of control methods

In this chapter two methods, namely Lyapunov's basic stability theory (Section 6.1) and model predictive control (Section 6.2), are further investigated for their suitability to deal with the control problems as defined in Section 5.2. In the last section of this chapter, the performance of the two control methods is summarized and the future of the control of PDE-models is discussed.

6.1 Lyapunov's basic stability theory

In order to investigate the stabilizability of the PDE-system (3.23) a Lyapunov function is required. This function, describing the 'energy' of the system, should be zero at the equilibrium point and positive at all other points. Several Lyapunov functions can be found for the considered PDE-system. Here, a relatively simple function is considered:

$$V(t) = \frac{4}{6\mu m} \int_0^1 \left((m + \rho(x,t))^3 - (m + \rho_{ss})^3 \right)^2 dx, \tag{6.1}$$

in which V is the Lyapunov function, μ is the process rate, m is the number of machines in the line and ρ is the density, which should be stabilized at the steady state value ρ_{ss}. It can be seen that (6.1) is positive for all ρ, except for $\rho = \rho_{ss}$. The term $4/(6\mu m)$ has been added to the function in order to simplify the time-derivative of (6.1):

$$\dot{V}(t) = -(m + \rho_1(t))^4 + (m + \rho_0(t))^4 + (m + \rho_{ss})^3(\rho_1(t) - \rho_0(t)). \tag{6.2}$$

The derivation of this equation can be found in Appendix A.2. Here, $\rho_0(t)$ is the density at the left boundary (the entrance of the system), which is coupled directly to the influx and thus can be seen as a control variable. The density at the right boundary $\rho_1(t)$ (the exit of the system) can not be used as a control variable. This makes the situation a little different from the one described by Coron et al. [Cor02], in which both boundaries can

be used for control. However, their idea to apply boundary control in order to influence the time-derivative of the Lyapunov function $\dot{V}$, can still be used.

In order to assure stabilization of the system, $\dot{V}$ should be negative (i.e., the 'energy' level should decrease) at all times until the steady state has been reached; then, $\dot{V}$ should be zero, so that the system remains in that state.

The question that remains is how to choose $\rho_0(t)$ in order to obtain a negative $\dot{V}$. Two parameters are available that can be used for feedback: the value of the Lyapunov function $V(t)$ and the density at the right boundary $\rho_1(t)$. Two propositions are made for $\rho_0(t)$:

1. $\rho_0(t) = \rho_1(t)$
2. $\rho_0(t) = \alpha(t) \cdot V(t) + \rho_1(t)$, for which $\alpha(t)$ is a scaling parameter.

The first proposition will be disregarded, since $\dot{V}$ now equals zero for all values of $\rho_1(t)$. It is however interesting to see that this option gives a stable dynamic solution: the 'energy' of the system remains at the same level. The second proposition, however, provides a solution: for steady state ($V = 0$), $\rho_0(t)$ equals $\rho_1(t)$ and thus $\dot{V} = 0$. For other values of V, a suitable value for α can be chosen, so that $\dot{V}$ is negative. It is even possible to find a value for α, for which $\dot{V}$ is maximally negative (and thus the 'energy' is maximally decreasing). To find this value, the second proposition for $\rho_0(t)$ is substituted in (6.2), and the derivative of $\dot{V}$ with respect to α is calculated:

$$\dot{V} = -(m + \rho_1)^4 + (m + \rho_1 + \alpha V)^4 - 4(m + \rho_{ss})^3 \alpha V, \tag{6.3}$$

$$\frac{\delta \dot{V}}{\delta \alpha} = 4V(m + \rho_1 + \alpha V)^3 - 4V(m + \rho_{ss})^3. \tag{6.4}$$

By setting (6.4) equal to zero it can be determined that $\dot{V}$ has a minimum for:

$$\alpha = \frac{\rho_{ss} - \rho_1}{V}. \tag{6.5}$$

Substituting (6.5) into the second proposition for ρ_0 shows that:

$$\rho_0(t) = \rho_{ss} \tag{6.6}$$

provides a minimum of $\dot{V}$. Further calculations indicate that for (6.6) the minimum of $\dot{V}$ is always negative, except for the case that $\rho_1(t) = \rho_{ss}$, for which $\dot{V} = 0$. In other words: for $\rho_0(t)$ as defined in (6.6), the system stabilizes at the required steady state in the shortest possible time. It is noted that the other Lyapunov functions that have been evaluated, come up with the same result.

Note that the solution in (6.6) does not use feedback data and that it corresponds to the common manufacturers' experience: „to get a system (rapidly) in steady state, just use the influx that corresponds to that steady state."

Backlog

Two of the problems introduced in Section 5.2 are concerned with the backlog that is often involved with a change of steady state. In order to solve these problem, first the *amount* of backlog should be investigated that is caused by a change of steady state. This is done using the following example:

Example 6.1 (The amount of backlog) *Consider an M/M/1 manufacturing line with 10 identical machines ($\mu = 2.0$). At time $t = 0$, the system is in steady state with an arrival rate of $\lambda_1 = 1.0$, so that all machines have a utilization of 50% and thus the amount of WIP in the line is (according to (4.1)):*

$$w_1 = 10 \cdot \frac{0.5}{1 - 0.5} = 10.$$

At $t = 0$, the customer demand equals the outflux and there is no backlog ($b_1 = 0$). For $t > 0$, however, the customer demand changes to 1.5. As a respons, the influx is directly changed to $\lambda_2 = 1.5$. At $t = t_{ss}$, the new steady state has been reached, the outflux equals the customer demand again and the backlog does not change anymore. The amount of WIP in the line is now:

$$w_2 = 10 \cdot \frac{0.75}{1 - 0.75} = 30.$$

The total amount of lots that entered the line in the time span $[0, t_{ss}]$ is $\lambda_2 t_{ss} = 1.5 t_{ss}$. The total customer demand in this time span is also $1.5 t_{ss}$. The number of finished lots in this time span can be calculated as follows:

$$\begin{aligned} \text{finished lots} &= \text{generated lots} + \text{WIP}(t = 0) - \text{WIP}(t = t_{ss}), \quad (6.7) \\ &= 1.5 t_{ss} + w_1 - w_2, \\ &= 1.5 t_{ss} + 10 - 30. \end{aligned}$$

The backlog thus has become:

$$\begin{aligned} b_2 &= b_1 + \text{total customer demand} - \text{finished lots}, \quad (6.8) \\ &= 0 + 1.5 t_{ss} - (1.5 t_{ss} + 10 - 30), \\ &= 20. \end{aligned}$$

From this example, two important properties can be learned:

- The amount of backlog that is formed during the transition of one steady state to another is independent of the duration of the transient state (t_{ss} in the example can have any value).

- The amount of backlog as mentioned in the previous item can easily be calculated using (6.7) and (6.8).

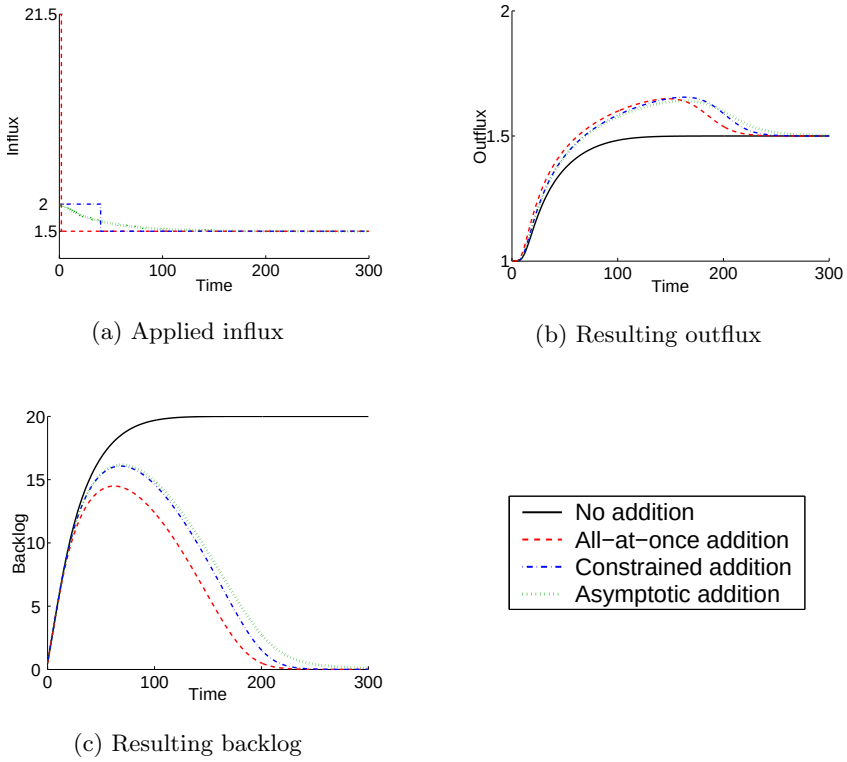
Once the amount of backlog caused by a transition between steady states is known, this amount should be added to the 'normal' arrival pattern (6.6) in order to bring the backlog back to zero. For positive backlog the influx is increased, while for negative backlog the influx is decreased. This adaption of the influx can be done in several ways, of which three will be evaluated for the system as presented in Example 6.1:

- *All-at-once* addition: the calculated amount of backlog is added as a whole to the influx in the first unit of time. In the rest of the time, the influx remains equal to the new steady state value d_2. Note that for a negative backlog, the influx can not be decreased further than 0, which restricts the use of this method.

- *Constrained* addition: the calculated amount of backlog is added gradually, such that the constraints that apply for the influx are respected. During the first units of time, the maximum allowable amount is added to the influx, until the entire amount of backlog has been added. Then, the influx is set at the new steady state value d_2. Here, the upper limit on the influx is set equal to the process rate, whereas the lower limit is set at zero.

- *Asymptotic* addition: each unit of time a fixed fraction (of the remainder) of the calculated amount of backlog is added to the influx. In this way, the influx changes gradually to its new steady state value d_2 as time passes by. Here, the value of the fixed fraction is chosen in such a way that the influx meets its constraint at $t = 0$.

In Figure 6.1 the influx and resulting outflux and backlog are shown as a function of time for the three methods of addition applied to the system as described in Example 6.1. From this figure, it can be read that the all-at-once method reaches stability faster than the other two methods. For the asymptotic method, the time to steady state can be shortened by increasing the value of the fraction. However, in this case the influx constraint would be exceeded. For the case that the fraction equals 1, the asymptotic method equals the all-at-once method. Note that by adapting the influx in the way it is described above, the assumed exponential distribution for the influx can be questioned. Especially for cases in which a large backlog should be annihilated, this might influence the reliability of results such as presented in Figure 6.1.



(a) Applied influx

(b) Resulting outflux

(c) Resulting backlog

Figure 6.1: Comparison of three 'backlog-removing' methods

Application to the control problems

With the derived controller (6.6), control problem 1 can be solved. In order to move the system from any initial state to a steady state corresponding to the outflux d_2, the influx should be set at d_2. Note that in this control problem the steady state is defined by the outflux, not by the density. However, since the flux is a function of the density, (6.6) can also be written in terms of flux, and thus it can be applied to this control problem. Results of this controller can be found in Appendix G (test results from Chapter 4), since they were obtained using the same input signals.

The second control problem can be solved using the controller (6.6) in combination with one of the 'backlog-removing' methods described in the previous subsection. It depends on the influx constraints what method can be used best. A typical result of this control method is visualized in Figure 6.1.

In order to solve control problem 3, the process rate μ should be written as a complex function of place and time. Even though this function can be formulated, a way to control the resulting PDE-model using Lyapunov's stability theorem has not yet been found.

6.2 Model predictive control

Before MPC can be applied to the control problems stated in Section 5.2, first two questions have to be answered:

- How can the PDE-model be discretized and what are the consequences of this discretization?
- What version of MPC should be used: linear MPC or nonlinear MPC?

These questions are answered in the following two subsections. Other information on MPC and its implementation can be found in Appendix H. In the last subsection, the application of MPC to the control problems is discussed. All visual results in this section are based on a 10-machine line, as introduced in Section 2.2, which is ramped up from an empty state to a steady state with a corresponding throughput of 1.5 lots per hour.

Discretization of the PDE-model

The PDE-model should be discretized with respect to place and time, before it can be used as the internal model of MPC. Recall that in order to solve the PDE-models, the fully discrete method (Appendix C.1) was used, which resulted in the C^{++}-code described in Appendix E. Here, a simplification of this discretization is used: the line is cut into 10 equal pieces of space (i.e., one piece for each machine) and the time is discretized according to:

$$dt \leq \frac{dx}{v_{\max}(t)}, \tag{6.9}$$

in order to guarantee stability of the PDE-model (Appendix C.1). Remember that for Model 3:

$$v_{\max} = \frac{\mu}{m + \rho^*}, \tag{6.10}$$

in which ρ^* is the minimal density at the current moment. In order to simplify the discretization with respect to time, the absolute maximum of v is taken:

$$v_{\max} = \frac{\mu}{m + 0} = \frac{\mu}{m}, \tag{6.11}$$

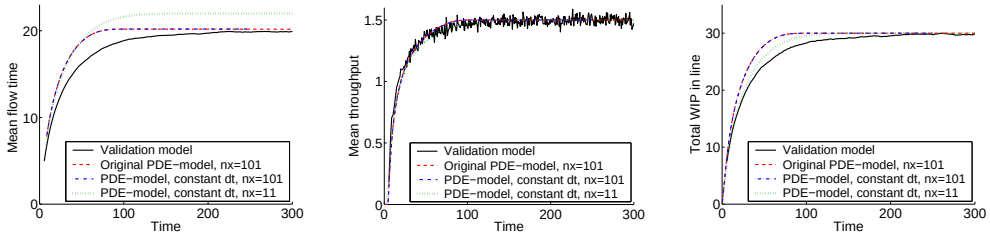
so that the condition for dt becomes:

$$dt = \mathit{cflcond} \cdot \frac{m \cdot dx}{\mu}, \tag{6.12}$$

in which $0 < \mathit{cflcond} \leq 1$. With the mentioned simplifications, the discretized PDE-model becomes:

$$\begin{aligned} \rho_1(t + dt) &= \rho_1(t) - \frac{dt}{dx}\left(\frac{\mu\rho_1(t)}{m + \rho_1(t)} - u(t)\right), \\ \rho_i(t + dt) &= \rho_i(t) - \frac{dt}{dx}\left(\frac{\mu\rho_i(t)}{m + \rho_i(t)} - \frac{\mu\rho_{i-1}(t)}{m + \rho_{i-1}(t)}\right), \qquad i = 2, 3, \ldots, 10. \end{aligned} \tag{6.13}$$

In Figure 6.2, the validation model and the original (C^{++}) PDE-model are visualized as well as the simplified discretizations that were proposed in the above text. For the green curve, only the discretization with respect to the time is simplified using (6.12). For the blue curve, in addition, the number of discrete points in the x-direction nx is reduced from 101 to 11.



(a) Mean flow time (b) Mean throughput (c) WIP

Figure 6.2: The influence of discretizing the PDE-model

Figure 6.2 shows that by applying (6.12), the behaviour of the PDE-model hardly changes. Decreasing nx, however, results in a considerable deviation of the steady state value of the mean flow time, while the steady state values of the mean throughput and mean WIP are not influenced. This influence of nx was already noted in Chapter 4 and further investigation is required to find a plausible explanation for this violation of Little's Law. For now, the simplified discretization of the PDE-model (6.13) is used as the internal model for MPC. Since MPC is meant to control the density of the system, and thus the flow time is not considered, the small value for nx will not deteriorate the performance of MPC. In fact, in Figure 6.2 it can be seen that the WIP is even described better for $nx = 11$ than for $nx = 101$.

Linear MPC versus nonlinear MPC

There are several ways in which model predictive control can deal with nonlinear model equations [Ess02b]. One way is by successive linearization: at each time sample, the model equations are linearized around the momentary working point and the resulting linear model is used for prediction and optimization over a future horizon. Because of its varying linear models, this method is often also referred to as linear time-variant MPC. Another possibility is to use nonlinear MPC: the nonlinear model equations are used directly in the internal model for prediction and optimization.

Although both versions of MPC have the same basic principles, an optimization problem for nonlinear MPC requires in general more computational effort than for linear MPC. For linear time-variant MPC, which uses a linear internal model and a goal function that is quadratic with respect to the design variable (i.e., the influx), the optimization problem is per definition convex. For this kind of optimization problems, several fast

and 'proven' algorithms are available. Nonlinear MPC, on the other hand, uses a nonlinear internal model and a goal function that is nonlinear (read: with a higher degree than quadratic) with respect to the design variable. Nonlinear MPC requires more computational effort because of the nonlinear calculations, but especially because the optimization problem usually looses its convex property and is therefore much harder to solve.

The linear time-variant MPC is however not very suitable to control the discretized PDE-model (6.13), because of the model's high degree of nonlinearity. Especially for situations in which the future has to be predicted over a long horizon, a single linearization per iteration, with which the whole prediction is made, will not do. Nonlinear MPC, on the other hand, might be suitable here. For this particular case, the nonlinear optimization problem seems to be convex: the existing constraints are all linear, and in case of only one design variable (i.e., one value for the influx that is constant over the prediction horizon), the optimization problem is convex. For the optimization problem with more design variables (i.e., the influx as a function of time), the convexity of the problem has not yet been proven, but, by simulation, results are obtained that indicate convexity of the problem.

It has been decided to use nonlinear MPC to solve the control problems. Since an implementation of nonlinear MPC was not yet available, it has been created in MATLAB® for this purpose. The implementation is based on an implementation of linear time-variant MPC [Ess02a] and can be found in Appendix H.

Application to the control problems

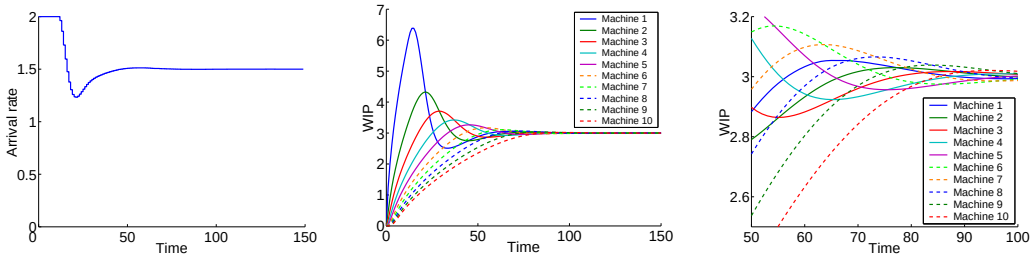
In order to solve a control problem with MPC, the objective of that problem should be defined in the goal function of the MPC implementation. A general form of this goal function is:

$$J(\mathbf{u}) = \sum_{i=1}^{p} \|(y(k+i) - y_{\text{ref}})\|_{\mathbf{Q}}^2 \,, \tag{6.14}$$

in which $\mathbf{u}$ is the vector of current and future inputs that should be optimized in order to minimize the goal function. Further, k is the current sample, p is the length of the prediction horizon, $y(k+i)$ is the output value at sample $k+i$, y_{ref} is the reference value for that output, and $\|x\|_{\mathbf{Q}}^2 = x^T\mathbf{Q}x$, in which $\mathbf{Q}$ is a weighting matrix. It is noted that the sample time considered here is a multiple of dt as defined in (6.12), but does not necessary need to be equal to dt.

With the implementation of nonlinear MPC as described in Appendix H, control problem 1 can be solved. For this purpose, the goal function that should be minimized equals (6.14), in which the input vector $\mathbf{u}$ represents the influx (constrained to the interval $[0, 1]$) and the output y represents the density in the last piece of the line (i.e., with the last machine). By moving this value to its reference value, the outflux of the

line can be controlled. In Figure 6.3 the optimized input signal is depicted as well as the resulting WIP-levels. In this figure it can be seen that the control signal is different from the constant input signal as proposed by the controller based on Lyapunov's stability theory, and that the system is stabilized at the desired steady state in a shorter time. Recall that a system is assumed to have reached its steady state if its outputs remain between 99% and 101% of the steady state value (Section 4.1).



(a) Input (b) WIP (c) WIP, zoomed in

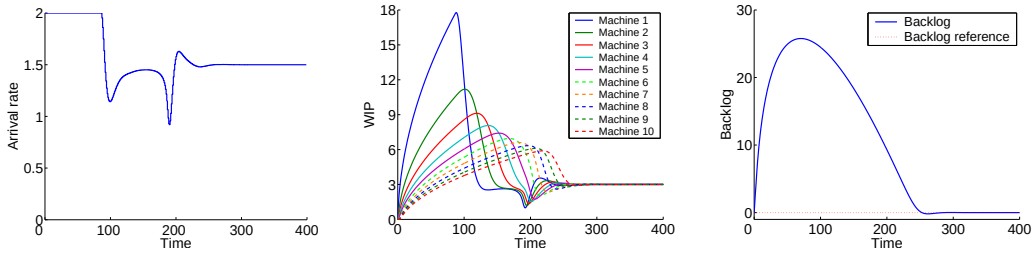
Figure 6.3: Nonlinear MPC solution for control problem 1

Since the results clearly show that the nl-MPC solver is capable of solving the first control problem, now control problem 2 is considered, which deals with the additional demand of preventing a permanent backlog. In order to solve this problem, the goal function is defined as a function of the backlog:

$$J(\mathbf{u}) = \sum_{i=1}^{p} \|(b(k+i))\|_{\mathbf{Q}}^2 , \tag{6.15}$$

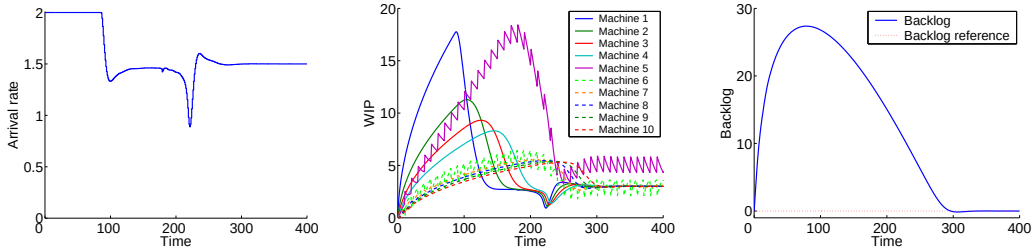
in which $b(k+i)$ is the backlog at future sample $k+i$. Note that once the backlog has stabilized at zero, this implies that the system has reached the correct steady state. Therefore, there is no need to include the density in the goal function. Solving this problem with nl-MPC results in the behaviour as depicted in Figure 6.4. In this figure, it can be seen that the backlog is indeed reduced to zero by a temporarily increase of the influx.

Now, control problem 3 is considered. Again, the backlog problem as defined in control problem 2 is considered, but this time for the case that one machine in the manufacturing line is subject to preventive maintenance. In order to solve this problem, the goal function of control problem 2 (6.15) is used and the definition of the system is changed (see Appendix H). In Figure 6.5, the optimized input is visualized, as well as the resulting WIP and backlog as a function of time. From these results, it can be concluded that the implementation of nl-MPC is able to solve control problem 3 as well.



(a) Input (b) WIP (c) Backlog

Figure 6.4: Nonlinear MPC solution for control problem 2



(a) Input (b) WIP (c) Backlog

Figure 6.5: Nonlinear MPC solution for control problem 3

6.3 Discussion

In this chapter, two control methods have been applied in order to control the PDE-model from its boundary. With Lyapunov's stability theory, a solution could be found for the first two control problems (Section 5.2). For this purpose, the continuous PDE-model was used in the definition of the Lyapunov function, the backlog was analyzed and quantified, and a stabilizing input was derived. The third control problem could not be solved using Lyapunov's stability theory, because of the complex system description that is caused by the preventive maintenance property. Model predictive control has been successfully applied to all three control problems. Before it could be applied, however, an implementation of nonlinear MPC had to be created in MATLAB®, with a discretization of the PDE-model as internal model. Through definition of a goal function and an internal model, the control problems were fed to the nl-MPC algorithm and an optimal solution was found.

An important conclusion that can be drawn from Lyapunov's stability theorem is that the considered system can be stabilized, independently on its (past) state, using a con-

stant input that is equal to the reference throughput. This implies that any (reasonably constrained) input signal is output-stabilizing as long as the input stabilizes at some point in time at the reference throughput value. Note that this knowledge has also been used to solve control problem 2: the input is first increased until the predetermined amount of backlog was compensated, after which the input is set equal to the reference throughput.

Although it appears from Lyapunov's stability theory that choosing the desired throughput as input results in the fastest stabilization, nl-MPC claims to have found an input signal that results in an even faster stabilization. Two possible explanations have been found for this apparent contradiction. One explanation is that there exists a more optimal input than the reference throughput, but that it can not be described with the structure $\rho_0(t) = \alpha(t) \cdot V(t) + \rho_1(t)$. Other structures for $\rho_0(t)$ with for example higher derivatives of $V(t)$ and $\rho_1(t)$ or other feedback parameters might lead to a more optimal solution. Another explanation is that the discretization of the PDE-model with respect to the place simplifies the model in such a way that it can be stabilized faster. More research is required to verify the correctness of these explanations.

So far, only a simple system and simple control problems have been considered. Lyapunov's stability theory was not sufficient to solve all three the control problems. Nonlinear MPC, on the other hand, was. It remains to be investigated to what extend the conclusions drawn from these simple problems can be used to solve larger and more complex (practical) problems. Furthermore, an interesting point of attention for future research is to find out to what extend the complexity of the system and/or control problems can be increased when using these control methods.

An extension that requires investigation, for example, is the robustness of nl-MPC. So far, the nl-MPC algorithm only solved control problems for which the internal model perfectly described the system to be controlled. Whether nl-MPC is robust enough to control the real-life system (represented by a DEM), with the discretized PDE-model as internal model, is investigated in Chapter 7.

Chapter 7

Controller application on a real-life system

Now that a controller has been designed, the final step of the control framework (Figure 1.1) can be considered: the coupling of the continuous controller to the real-life manufacturing system (represented by a DEM). In the previous chapter, two control methods have been considered: Lyapunov's stability theory and nl-MPC. Since a Lyapunov controller does not use feedback, it is not useful to apply in the control framework. Therefore, only the nl-MPC control method is considered here.

In order to couple the controller to the DEM, two conversions have to be made. Section 7.1 gives a short introduction into the field of converters, but a lot of research remains to be done here. After two basic converters have been designed, the controller is coupled to the DEM in Section 7.2 and a simple control problem is considered. In Section 7.3, this experiment is evaluated and future research is discussed.

7.1 Design of converters

The two converters that need to be designed for the control framework, are of a different nature. The first converter needs to transform the discrete-time input from the controller into an exponentially distributed arrival rate for the DEM. The second converter is in fact an observer that has to filter out suitable data for the controller from the output signals of the DEM. The first converter is addressed further as *input converter*, the second converter as *output filter*. For each of these converters, the demands are now presented and the design is discussed.

Input converter

An important demand for the input converter is that it does not contain any form of control or dynamics, since this would affect the performance of the controller (the controller is designed for the dynamics of the manufacturing system only). This demand means, for example, that data from the past may not influence the current conversion, since such a memory function implies dynamics.

Because of the discrete event nature, the conversion can not take place continuously. Therefore, it is performed in periods: for each period the input signal from the controller is considered and a corresponding arrival rate for the DEM is determined. The sample time of the controller is assumed to equal the length of a period, i.e., the input signal is constant during a period. Two questions that now remain to be answered are:

- How should the arrival pattern of the DEM be defined?
- What is a suitable sample time?

There are several possible answers to the first question. For example, in each period, the lots that should be generated can be fed all at once to the buffer of the first machine. Another option is to feed the lots deterministically to the first buffer with the rate defined by the controller. For these options, the average of generated lots per period is (almost) exact, but the arrival rate is far from exponentially distributed. Since the M/M/1 property of the line is important for the accuracy of the approximation model, and thus for the controller, these options are not acceptable. A better solution is to feed, in each period, the new input of the controller to the DEM as the new mean value of the negative exponential distribution that describes the arrival rate. In this way, the average of generated lots per period often is not exact (because of the exponential distribution), but at least the M/M/1 property is more accurately observed. This conversion method is therefore used for the input converter.

In order to get a good conversion, it is important that besides this M/M/1 property also the arrival *rate* is observed as good as possible. Although the mean value of the distribution is set at the desired rate, it is only after a significant number of samples that this mean value becomes clear. Therefore, the length of a period should be defined in such a way that it contains enough samples. On the other hand, the period (= sample time of controller) should not be chosen too long, since this restricts the freedom of the controller, and thus its performance. Although further research is required to find an 'optimal' value for the period, here it is assumed that the period is long enough when it contains an average of at least 10 samples. Using the steady state throughput as reference for the arrival rate, the minimal length of a period can be determined:

$$\text{period length} \geq 10\frac{1}{\delta_{ss}}. \tag{7.1}$$

Output filter

The measured output of the DEM, is the WIP at the last machine in the line. Since only the average value of this highly stochastic variable is of interest to the controller, this value should be filtered out by the output filter. A second task of the output filter is to compensate for the errors of the internal approximation model of the nl-MPC controller, in order to obtain a more robust controller. Both tasks of the output filter are performed by a first order non-model based observer. Before this observer is discussed, however, first the functions and types of filters are examined[Ess02b].

Generally, observers (also referred to as filters) are used for three purposes:

- *State reconstruction*: if not all elements of a state can be measured, an observer is required to estimate or reconstruct the missing elements.
- *Filtering*: if a signal is disturbed by noise, an observer can be used to filter out the undisturbed signal.
- *Correction*: if an internal model does not predict the system behaviour well due to mismatches in parameters or unmodelled phenomena, an observer is required to correct these errors.

Basically, two kinds of observers can be distinguished: non-model based observers and model based observers. For the non-model based observer, the system's state is augmented with an extra variable that describes the disturbance d:

$$d = y_{\text{meas}} - y, \tag{7.2}$$

in which y_{meas} is the measured output and y is the output predicted by the internal model. The model output is now defined as a function of the state and the disturbance parameter:

$$y(k) = h\left(x(k)\right) + d(k). \tag{7.3}$$

For a constant output disturbance observer, the value of d is assumed to be constant. For a first order output disturbance observer, or shortly first order observer, the value of the disturbance parameter is adapted by:

$$\begin{aligned} d(k+1) &= d(k) + K_d\big(y_{\text{meas}}(k) - y(k)\big), \\ &= d(k) + K_d\big(y_{\text{meas}}(k) - h\left(x(k)\right) - d(k)\big), \end{aligned} \tag{7.4}$$

in which K_d is the filter gain that determines the rate of convergence of the disturbance parameter. The new description of the internal model becomes:

$$\begin{aligned} \begin{bmatrix} x(k+1) \\ d(k+1) \end{bmatrix} &= \begin{bmatrix} f(x(k), u(k)) \\ d(k) + K_d\big(y_{\text{meas}}(k) - h\left(x(k)\right) - d(k)\big) \end{bmatrix}, \\ y(k+1) &= h\left(x(k+1)\right) + d(k+1). \end{aligned} \tag{7.5}$$

The model based observer does not add a variable to the system's state. Instead, it adapts the state of the internal model directly, such that the output of the model corresponds to the output of the real system. The classical state observer is the Kalman filter. Since a Kalman filter for a nonlinear model is however very complicated, this observer is not considered here.

The first order non-model based observer is applied to obtain a mean value for the disturbance parameter from the stochastic output signal of the DEM (the stochastic signal can be seen as a noise around the mean WIP value), using (7.4). In order to get an accurate disturbance parameter for each sample of the controller, the sample frequency of the observer should be much higher than the sample frequency of the controller. On the other hand, it should not be too high, since this would require more computer effort, without gaining extra data (the WIP would remain constant during several samples). Furthermore, the filter gain should be very small, so that each individual measurement on the DEM hardly affects the disturbance parameter and that only long-term deviations in the WIP are monitored in the disturbance parameter. On the other hand, if it is too small, the measurements do not affect the disturbance parameter at all, and the observer has no effect.

7.2 Control of the real-life system

Now that an input converter and an output filter have been selected. The controller can be coupled to the DEM. In Figure 7.1 the resulting framework is visualized.

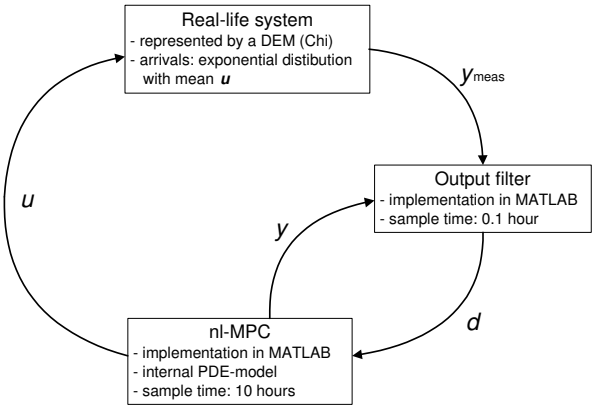
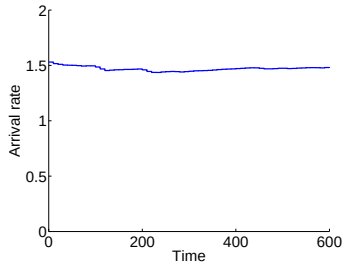


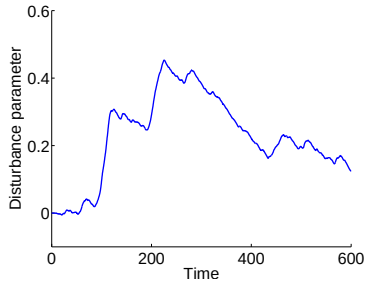
Figure 7.1: Control framework for the real-life system

In order to test the performance of this control framework, the first control problem of Section 5.2 is considered for the 10-machine manufacturing line described in Section 2.2: the empty system should be ramped up in the shortest possible time to a steady state that corresponds to a throughput of 1.5 lots per hour. Here the sample time of the controller is set at 10 hours, the sample time of the observer is set at 0.1 hour and the

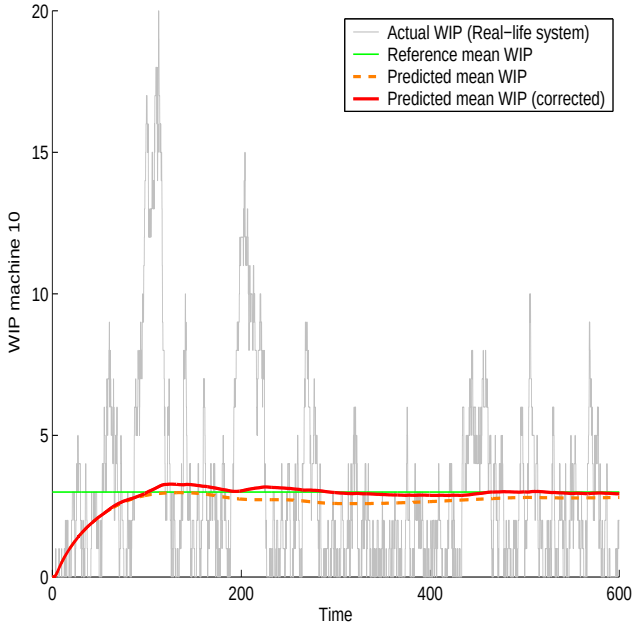
filter gain K_d is set at 10^{-4}. The implementation of this control framework is described in Appendix I.



(a) Input u



(b) Disturbance parameter d



(c) WIP w

Figure 7.2: Simulation results of the controlled DEM

In Figure 7.2 the results are visualized that were obtained by applying the control framework to the DEM. The figures show that the converters and the controller perform well enough to stabilize the DEM at the desired steady state. Figure 7.2(a) illustrates that the determined control input is less aggressive than the one in Figure 6.3(a). This is a result of the larger sample time. In Figure 7.2(c) the output of the real-life system and the (corrected) output predicted by the internal model are plotted. Here, it can be seen that the output of the real-life system is a noisy (stochastic) signal and that the output of the internal model is corrected with the disturbance parameter (Figure 7.2(b)) in order to predict a good average of the real system's output. Note that the development of the disturbance parameter is only characteristic for this simulation. Because of the stochastic property of the DEM, another simulation with identical parameters would result in a totally different output and therefore a totally different disturbance parameter curve.

7.3 Discussion

It has been shown in this chapter that the considered DEM can be controlled by a 'continuous' nl-MPC controller, with the use of two converters. Note, however, that in order to so, the sample time of the controller must be increased significantly, which has a negative effect on the performance of the controller. In Chapter 6, nl-MPC was able to optimize the input signal in such a way that the system reached stability faster than it would do with a constant input that equals the desired throughput (Lyapunov's stability theory). The nl-MPC controller with the larger sample time as considered in this chapter, on the other hand, proposes an input signal that almost equals the constant input signal. It can therefore be concluded that nl-MPC only performs better than the Lyapunov controller, if the sample time of the controller is small enough.

It should be investigated whether an input conversion can be designed that has no need for an increase of the sample time of the controller. If it can be designed, the performance of the nl-MPC controller can improve. If, on the other hand, such a design can not be developed or causes other performance problems, the use of nl-MPC should be reconsidered. For control problems that can be solved cheaply with for example Lyapunov's stability theory, the use of nl-MPC is a waste of computational effort if it does not provide extra performance. An advantage of nl-MPC above other controllers such as the Lyapunov controller is, however, that by means of the feedback it can react for example on situations in which the stochastic behaviour gets out of hand.

Another way to improve the performance of the control framework (Figure 7.1), is to increase the accuracy of the predictions made by the internal model. This can be achieved, of course, by improvement of the PDE-model, which has already been a point of discussion in Chapter 4. An alternative, however, is to increase the performance of the observer. A better observer leads to a better correction of the internal model and therefore to improved accuracy of the predictions. Improvement of the output observer, for example by optimization the first order non-model based observer or application of a different (model based) observer, is left for future research.

Finally, a point of investigation that should be considered, is the comparison of the performance of this control framework with that of other control strategies for manufacturing lines. Based on the results as presented in this report, it is assumed that for simple control problems, the performance of the presented control framework is similar to that of other control strategies. For more complex problems, however, this control framework is assumed to outperform other control strategies because of its approximation model and feedback property. Further investigation is however required to validate these assumptions.

Chapter 8

Conclusions

For the control of manufacturing systems, recently a framework has been developed that makes use of an approximation model of the manufacturing system and a continuous controller. For a proper and useful functioning of this framework, it is important that the approximation model is continuous, provides a good description of flow time and throughput in both transient and steady state, and is computationally feasible. None of the 'established' approximation models for manufacturing systems, however, satisfy all these demands. It is only recently that the use of PDEs in the description of manufacturing systems has been considered. Although these PDE-models are continuous, their behaviour in the field of manufacturing has not been validated so far. In this report, therefore, the applicability of the PDE-models in the modelling and control of manufacturing systems has been investigated.

Validation of the PDE-models

Using a DEM as validation model, the performance of the three currently available PDE-models has been examined. The case that was used for this validation study is a flow line, consisting of identical M/M/1 processes. With this flow line, experiments have been performed that include both ramp up and ramp down simulations. For these experiments, the discrete event simulations were performed using the specification language χ, whereas the PDE-models were solved numerically using the fully discrete method.

In order to validate the performance of the PDE-models, parameters such as mean flow time, mean throughput and mean WIP have been investigated. In order to obtain these parameters, the simulation output of both the DEM and the PDE-models had to be processed. For the DEM, the mean values were filtered from the stochastic simulation output, whereas for the PDE-models, the parameters had to be derived from the flux and density data that were gathered during the simulation.

The results of this validation study show that the PDE-models accurately describe the steady state behaviour of a manufacturing system. The transient behaviour, however, leaves much to be desired. In addition to this, one of the PDE-models shows undesirable behaviour: lots that enter the system influence the velocity of lots further downstream in the system. The other two PDE-models do not show this behaviour and are comparable in their overall performance. Of these two models, the most simple model is selected for use in the remainder of this research.

Boundary control for the PDE-model

For the control of the selected PDE-model, several control methods have been investigated. Only the methods that are capable of dealing with a boundary control problem for PDEs were accepted, since it was defined that the PDE-model should be controlled using the influx only. The two most promising control strategies, Lyapunov's stability theory and nl-MPC, have been investigated further considering three control problems that are relevant for the manufacturing industry. These problems include time optimal switching between steady states, preventing permanent backlogs and taking into account system disturbances, such as preventive maintenance.

Lyapunov's stability theory resulted in a simple but time optimal controller for the stabilization of a system at a (different) steady state: one can best feed the manufacturing system with an influx (arrival rate) that corresponds to the desired steady state flux (throughput). Note that this control strategy is often used in practice and does not use any form of feedback. Furthermore, it turned out that a permanent backlog — due to the switching to another steady state — can be prevented easily. Finally, no method has been found so far to take into account preventive maintenance explicitly, when using Lyapunov's stability theory.

Before nl-MPC could be applied to solve the control problems, first an implementation of nl-MPC was created with a fully discretized version of the PDE-model as internal model. With this implementation, all three control problems have been solved. Unexpectedly, nl-MPC came up with a faster controller for the switch to another steady state than the already time optimal controller, which was derived with Lyapunov's stability theory.

Since the controller that was derived with Lyapunov's stability theory does not require feedback, it can be applied without the use of the control framework. Therefore, only nl-MPC is considered in the final part of this research.

Coupling the controller to a real manufacturing system

The last step in the control framework is to couple the designed continuous controller to the real manufacturing system, which is represented in this research by a DEM. For this purpose, two converters have been designed: one that converts the continuous output

of the controller into a suitable input signal for the DEM and one that converts the output of the DEM into suitable data for the controller.

The input converter, which links the controller signal to the arrival rate of the DEM, is simple: for each time sample, the influx as determined by the controller is used as the mean of the exponentially distributed arrival rate. However, in order to obtain a good correspondence between the control signal and the mean arrival rate, the mentioned sample time should be relatively large, which reduces the performance of the controller.

For the output converter, a first order non-model based observer is used, which filters out the disturbance (stochastic behaviour) of the DEM's output and provides mean values of the WIP to the controller. In order to get a good approximation of the mean WIP, the sample time of this observer have been set a factor 100 smaller than the sample time of the controller and a very small filter gain is used, so that each individual measurement does not influence the mean value too much.

With these converters, the nl-MPC controller is successfully coupled to the DEM. To test this combination of DEM, nl-MPC controller and converters, a simple ramp up experiment has been performed. Due to the longer sample time of the controller, the resulting control signal was less 'aggressive' and very close to the constant control signal that was predicted by Lyapunov's stability theory.

Chapter 9

Recommendations

During this research project, several questions have come up that could not be solved due to the limited duration of the project. For each main topic in this report, the most important questions are treated here. Based on these questions, recommendations for future research are formulated.

Performance of the PDE-models

The validation study showed that the considered PDE-models are not accurate at describing the transient behaviour of a manufacturing system. The cause of this deviation is however not clear. It is therefore recommended that this problem is investigated further and that the search for better PDE-models continues. Model 2, for example, might be improved by adding a term to the velocity-PDE, so that the velocity becomes more dependent on the density. Model 3 might be improved by adding a (one-directional) diffusion term to the PDE.

During the validation experiments it has also been observed that the performance of the PDE-models depends on the model parameters, such as the number of machines in the line, or the number of samples in x-direction. The number of experiments was too small, however, to determine the true relation between those parameters and the performance. More experiments should be performed to find these relations. This would also provide a better insight in the behaviour of the PDE-models, which is valuable in the search for better PDE-models.

So far, the validation study has only been performed for the flow line with identical M/M/1 processes. However, in order to pass a general judgement on the performance of PDE-models in the modelling of manufacturing system, more systems should be considered, such as re-entrant systems and systems with G/G/1 processes. In order to describe the latter, Model 2 might for example be extended with a third PDE that describes variability, such as described by [Hoo00]. Model 3, which is now based on the

queueing relations for M/M/1 processes, might be improved by using queueing relations for G/G/1 processes instead.

Another point of attention for future research is the method to solve the PDEs. During this investigation, the fully discrete method has been used, which adds (under some conditions) artificial diffusion to the models. Because of this artificial diffusion, the numerical approximation of the models can describe significantly different behaviour than the PDE-model itself would do. An obvious example of this difference can be seen for Model 3: the model itself is unable to describe a ramp down, whereas the numerical approximation has no problem with it at all. It should be investigated how this problem can be solved or whether other solution methods can be used instead.

Control of PDE-models

Although several boundary control methods for PDE-models have been investigated, some others are still unexplored, such as H^∞-optimal boundary control and Hamiltonian boundary control. Since these control methods might provide new insight in the control of manufacturing PDE-models and might come up with better controllers, it is suggested that they are investigated as well in future research.

Besides other control methods, different control problems should be investigated as well. In the manufacturing industry, there are more control problems than the ones described in this report. These problems should be identified and it should be determined whether they can be solved for the PDE-models using the currently available control methods. An example of a control problem that requires more research is the control of a manufacturing supply chain, i.e., a network of manufacturing systems.

Performance of the control framework

As the results on the performance of the control framework show, the considered control method is less efficient in the control framework than it is when applied to the continuous approximation model. Of course, a part of this loss can be attributed to the difference between the actual system and its approximation model. The other part of the loss, however, is caused by the conversions that take place in the framework. The input converter, which is to blame for the larger sample time of the controller and thus for a loss in performance, should therefore be a subject of future research.

Besides the fact that the performance of the control framework might be improved, it is also important to know how well this framework (with the PDE-model as approximation model) performs in comparison with other control strategies, so that a good estimation of its usefulness can be made. Therefore, it is recommended that the performance of this framework is compared with, for example, the simple heuristics that are currently used in manufacturing systems: what are the advantages or extra capabilities of this framework that other strategies do not have?

Bibliography

[Arm02a] D. Armbruster, D. Marthaler, and C. Ringhofer. Efficient simulations of supply chain. *Proceedings of the Winter Simulation Conference*, pages 1345–1348, 2002.

[Arm02b] D. Armbruster, D. Marthaler, and C. Ringhofer. A mesoscopic approach to the simulation of semiconductor supply chains. *Proceedings of the International Conference on Modeling and Analysis of Semiconductor Manufacturing (MASM2002)*, pages 365–369, 2002.

[Arm02c] D. Armbruster, D. Marthaler, and C. Ringhofer. Modeling a re-entrant factory. submitted, Operations Research, preprint available at `http://math.la.asu.edu/~chris`, 2002.

[Aw00] A. Aw and M. Rascle. Resurrection of 'second order' models of traffic flow. *SIAM Journal on Applied Mathematics*, 60(3):916–938, 2000.

[Baş98] T. Başar and M.Q. Xiao. H^∞-optimal boundary control of hyperbolic systems with sampled measurements. *Proceedings of the 37th IEEE Conference on Decision and Control*, 3:2830–2835, 1998.

[Bro01] J.L. Brown. Modal decomposition of convection-reaction-diffusion system. Master's thesis, University of Alberta, 2001.

[Buz93] J.A. Buzacott and J.G. Shantikumar. *Stochastic Models of Manufacturing Systems*. Prentice Hall, Englewood Cliffs, 1993.

[Cas02] C.G. Cassandras, Y. Wardi, B. Melamed, G. Sun, and C. Panayiotou. Perturbation analysis for on-line control and optimization of stochastic fluid models. *IEEE Transactions on Automatic Control*, 47(8):1234–1248, 2002.

[Chr96] P.D. Christofides and P. Daoutidis. Feedback control of hyperbolic PDE systems. *American Institute of Chemical Engineers Journal*, 42(11):3063–3086, 1996.

[Chr98] P.D. Christofides and P. Daoutidis. Robust control of hyperbolic PDE systems. *Chemical Engineering Science*, 53(1):85–105, 1998.

[Cor02] J.M. Coron, J. de Halleux, G. Bastin, and B. d'Andréa Novel. On boundary control design for quasilinear hyperbolic systems with entropies as Lyapunov functions. *Proceedings of the 41st IEEE Conference on Decision and Control*, 3:3010–3014, 2002.

[Dag95] C.F. Daganzo. Requiem for second-order fluid approximations of traffic flow. *Transportation Research*, 29B(4):277–286, 1995.

[Ess02a] H.A. van Essen. MATLAB®-implementation of linear time-variant MPC, 2002. Course material for the course 'Capita Selecta in Control', Eindhoven University of Technology, department of Mechanical Engineering.

[Ess02b] H.A. van Essen and M. Steinbuch. Lecture notes on model predictive control for the course 'Capita Selecta in Control', Eindhoven University of Technology, department of Mechanical Engineering, 2002.

[Gun98] P.K. Gundepudi and J.C. Friedly. Velocity control of hyperbolic partial differential equation systems with single characteristic variable. *Chemical Engineering Science*, 53(24):4055–4072, 1998.

[Har95] J.M. Harrison. *Brownian Motion and Stochastic Flow Systems*. John Wiley & Sons, Inc., New York, 1995.

[Hea97] M.T. Heath. *Scientific Computing: An Introductory Survey*. WCB/McGraw-Hill, Boston, 1997.

[Hof02] A.T. Hofkamp and J.E. Rooda. χ *Reference Manual*. Systems Engineering Group, Eindhoven University of Technology, 2002. `http://se.wtb.tue.nl`.

[Hof03] A.T. Hofkamp. *Python from* χ. Systems Engineering Group, Eindhoven University of Technology, 2003. `http://se.wtb.tue.nl`.

[Hoo00] S.P. Hoogendoorn and P.H.L. Bovy. Continuum modeling of multiclass traffic flow. *Transportation Research*, 34B:123–146, 2000.

[Hop00] W.J. Hopp and M.L. Spearman. *Factory physics: Foundations of manufacturing management*. Irwin/McGraw-Hill International Editions, Singapore, second edition, 2000.

[Jo03] T.-C. Jo. C^{++} code, 2003. Model for numerically solving PDE-models of manufacturing systems. Personal communication.

[Khi32] A. Khinchin. Mathematical theory of stationary queues. *Sbornik Mathematics*, 39:73–84, 1932.

[Kim83] J. Kimemia and S.B. Gershwin. An algorithm for the computer control of a flexible manufacturing system. *IIE Transactions*, 15(4):353–362, 1983.

[Law00] A.M. Law and W.D. Kelton. *Simulation modeling and analysis*. McGraw-Hill Higher Education, Boston, third edition, 2000.

[Lax72] P.D. Lax. The formation and decay of shock waves. *American Mathematical Monthly*, 79:227–241, 1972.

[Lee87] H.G. Lee, A. Arapostathis, and S.I. Marcus. Linearization of discrete-time systems. *International Journal of Control*, 45(5):1803–1822, 1987.

[Lef03] E. Lefeber. Nonlinear models for control of manufacturing systems. *Proceedings of the 4th International Symposium Investigations of Non-Linear Dynamic Effects in Production Systems*, Chemnitz, Germany, April 2003.

[Lig55] M.J. Lighthill and J.B. Whitham. On kinematic waves. I: Flow movement in long rivers. II: A theory of traffic flow on long crowded roads. *Proceedings of the Royal Society A*, 229:281–345, 1955.

[Lit61] J.D.C. Little. A proof of the queuing formula $l = \lambda w$. *Operations Research*, 9:383–387, 1961.

[Lya92] A.M. Lyapunov. *The General Problem of the Stability of Motion*. Taylor & Francis, London/Washington, 1992. Translation of the 1892 memoir by A.T. Fuller.

[McO03] R.C. McOwen. *Partial Differential Equations: Methods and Applications*. Prentice Hall, Upper Saddle River, second edition, 2003.

[Mon99] D.C. Montgomery and G.C. Runger. *Applied Statistics and Probability for Engineers*. John Wiley & Sons, Inc., New York, second edition, 1999.

[Plo03] H. Ploegmakers. Feedback control compared to common control strategies for a multi-product flow line. Master's thesis, Eindhoven University of Technology, 2003.

[Pol30] F. Pollaczek. Uber eine Aufgabe der Wahrscheinlichkeitstheorie. *I-II Mathematische Zeitschrift*, 32:64–100, 729–750, 1930.

[Rab99] R. Rabenstein and L. Trautmann. Solution of vector partial differential equations by transfer function models. *Proceedings on the 1999 IEEE International Symposium on Circuits and Systems*, 5:21–24, 1999.

[Ram87] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete-event systems. *SIAM Journal on Control and Optimization*, 25:206–230, 1987.

[Rem03] B. Rem. Control theory applied to a re-entrant manufacturing line. Master's thesis, Eindhoven University of Technology, 2003.

[Ric56] P.I. Richards. Shockwaves on the highway. *Operations Research*, 4:42–51, 1956.

[Sas99] S.S. Sastry. *Nonlinear Systems: Analysis, Stability, and Control*. Springer-Verlag, New York, 1999.

[Sch01] A.J. van der Schaft and B.M. Maschke. Fluid dynamical systems as Hamiltonian boundary control system. *Proceedings of the 40th IEEE Conference on Decision and Control*, 5:4497–4502, 2001.

[Sir89] H. Sira-Ramirez. Distributed sliding mode control in systems described by quasilinear partial differential equations. *Systems and Control Letters*, 13:177–181, 1989.

[Sur98] R. Suri. *Quick Response Manufacturing: A Companywide Approach to Reducing Lead Times*. Productivity Press, Portland, 1998.

[Wan66] P.K.C. Wang. Asymptotic stability of distributed parameter system with feedback controls. *IEEE Transactions on Automatic Control*, 11(1):46–54, 1966.

[Zac86] E.C. Zachmanoglou and D.W. Thoe. *Introduction to Partial Differential Equations with Applications*. Dover Publications, Inc., New York, 1986. Reprint. Originally published: Baltimore: Williams & Wilkins, 1976.

[Zau89] E. Zauderer. *Partial Differential Equations of Applied Mathematics*. John Wiley & Sons, Inc., Singapore, second edition, 1989.

Appendix A

Mathematical proofs and derivations

A.1 Feedback linearization for the PDE-model

Since feedback linearization is only applicable to systems of ODEs or difference equations, the PDE-model is discretized first. It is noted that the detail in the (x,t)-grid of the discretized PDE-model does not influence the feedback linearizablility. Therefore, in order to keep this proof of feedback linearizability simple, the discretized PDE-model is assumed to have only 3 state variables:

$$\begin{bmatrix} \rho_1(t+1) \\ \rho_2(t+1) \\ \rho_3(t+1) \end{bmatrix} = f(\rho(t),u(t)) = \begin{bmatrix} \rho_1(t) - \frac{dt}{dx}\left(\frac{\mu\rho_1(t)}{m+\rho_1(t)} - u(t)\right) \\ \rho_2(t) - \frac{dt}{dx}\left(\frac{\mu\rho_2(t)}{m+\rho_2(t)} - \frac{\mu\rho_1(t)}{m+\rho_1(t)}\right) \\ \rho_3(t) - \frac{dt}{dx}\left(\frac{\mu\rho_3(t)}{m+\rho_3(t)} - \frac{\mu\rho_2(t)}{m+\rho_2(t)}\right) \end{bmatrix}, \tag{A.1}$$

$$y(t) = h\cdot\rho(t) = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}\rho(t) = \rho_3(t).$$

According to [Lee87], feedback linearization of the input-output relation of system (A.1) is possible if and only if the system satisfies the following conditions:

1. $\left\{\left(\frac{\partial f}{\partial u}\right)_{(0,0)}, \left(\frac{\partial f}{\partial \rho}\right)_{(0,0)}\left(\frac{\partial f}{\partial u}\right)_{(0,0)}, \left(\frac{\partial f}{\partial \rho}\right)^2_{(0,0)}\left(\frac{\partial f}{\partial u}\right)_{(0,0)}\right\}$ are linearly independent.
2. $\left(\frac{\partial}{\partial u}(h\cdot f^r)\right)_{(0,0)} \neq 0$. Here, r is the characteristic number of f, i.e., the smallest integer for which $\partial/\partial u(h\cdot f^r) \not\equiv 0$. For the discretized system (A.1), $r=3$.
3. $\tilde{f}^i_*\left(\frac{\partial}{\partial u^*_1}\right)$ is a well-defined smooth vector field on an open neighbourhood of $0\in\mathbb{R}^3$ for $1\leq i\leq 4$, where $\tilde{f}(\rho,u^*) = f(\rho,g(\rho,u^*))$ and $h\cdot f^r(\rho,g(\rho,u^*)) = u^*$.

First the derivatives $\frac{\partial f}{\partial u}$ and $\frac{\partial f}{\partial \rho}$ are calculated:

$$\left(\frac{\partial f}{\partial u}\right)_{(0,0)} = \begin{bmatrix} \frac{dt}{dx} \\ 0 \\ 0 \end{bmatrix}, \qquad \left(\frac{\partial f}{\partial \rho}\right)_{(0,0)} = \begin{bmatrix} 1-\frac{dt}{dx}\frac{\mu}{m} & 0 & 0 \\ \frac{dt}{dx}\frac{\mu}{m} & 1-\frac{dt}{dx}\frac{\mu}{m} & 0 \\ 0 & \frac{dt}{dx}\frac{\mu}{m} & 1-\frac{dt}{dx}\frac{\mu}{m} \end{bmatrix}. \tag{A.2}$$

Now the first condition can be tested:

$$\left\{ (\frac{\partial f}{\partial u})_{(0,0)}, (\frac{\partial f}{\partial \rho})_{(0,0)}(\frac{\partial f}{\partial u})_{(0,0)}, (\frac{\partial f}{\partial \rho})^2_{(0,0)}(\frac{\partial f}{\partial u})_{(0,0)} \right\} = \tag{A.3}$$

$$\left\{ \begin{bmatrix} \frac{dt}{dx} \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} (1-\frac{dt}{dx}\frac{\mu}{m})\frac{dt}{dx} \\ \left(\frac{dt}{dx}\right)^2\frac{\mu}{m} \\ 0 \end{bmatrix}, \begin{bmatrix} (1-\frac{dt}{dx}\frac{\mu}{m})^2\frac{dt}{dx} \\ 2\left(\frac{dt}{dx}\right)^2\frac{\mu}{m}(1-\frac{dt}{dx}\frac{\mu}{m}) \\ \left(\frac{dt}{dx}\right)^3\left(\frac{\mu}{m}\right)^2 \end{bmatrix} \right\}$$

has full rank, so condition 1 is satisfied. In order to test the second condition, the following calculations are made:

$$\begin{aligned} \frac{\partial}{\partial u}(h\cdot f^1) &= \frac{\partial}{\partial u}(\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}\cdot f(\rho(t),u(t)) = \frac{\partial \rho_3(t+1)}{\partial u} = 0, \\ \frac{\partial}{\partial u}(h\cdot f^2) &= \frac{\partial}{\partial u}(\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}\cdot f(f(\rho(t),u(t)),u(t+1)) = \frac{\partial \rho_3(t+2)}{\partial u} = 0, \\ \frac{\partial}{\partial u}(h\cdot f^3) &= \frac{\partial}{\partial u}(\begin{bmatrix} 0 & 0 & 1 \end{bmatrix}\cdot f(f(f(\rho(t),u(t)),u(t+1),u(t+2))) = \frac{\partial \rho_3(t+3)}{\partial u} \neq 0. \end{aligned} \tag{A.4}$$

It can be seen that $h\cdot f^3$ is dependent on u, since $\rho_3(t+3)$ is dependent on $\rho_2(t+2)$ and thus on $\rho_1(t+1)$, in which the input appears. The last equation of (A.4) can be worked out to find that $\frac{\partial}{\partial u}(h\cdot f^3)_{(0,0)} \neq 0$, and thus condition 2 is satisfied.

Before the third condition can be tested, first a state-coordinate change is performed and the feedback law

$$u^* = h\cdot f^3(\rho,u) = \rho_3(t+3) \tag{A.5}$$

is applied:

$$\begin{bmatrix} z1(t) \\ z2(t) \\ z3(t) \end{bmatrix} = \begin{bmatrix} h\cdot\rho(t) \\ h\cdot f(\rho,0) \\ h\cdot f^2(\rho(t)) \end{bmatrix} = \begin{bmatrix} \rho_3(t) \\ \rho_3(t+1) \\ \rho_3(t+2) \end{bmatrix}, \tag{A.6}$$

$$\begin{bmatrix} z1(t+1) \\ z2(t+1) \\ z3(t+1) \end{bmatrix} = \begin{bmatrix} z2(t) \\ z3(t) \\ u^* \end{bmatrix} = \tilde{f}(z(t),u^*). \tag{A.7}$$

According to [Lee87], $\tilde{f}_*^i\left(\frac{\partial}{\partial u_1^*}\right)$ is a well-defined smooth vector field for $1\leq i\leq 4$, if $[\partial/\partial u_i^*, \ker(\mathcal{F}_0)_*] \subset \ker(\mathcal{F}_0)_*$ for $1\leq i\leq 4$, in which the [ ] denote Lie-brackets. Here,

$$\mathcal{F}_0 = \tilde{f}(\tilde{f}(\tilde{f}(\tilde{f}(0,u_4^*),u_3^*),u_2^*),u_1^*) = \begin{bmatrix} u_3^* \\ u_2^* \\ u_1^* \end{bmatrix}. \tag{A.8}$$

Furthermore, $(\mathcal{F}_0)_*$ is the surjective (i.e., 'Hamiltonian') of $\mathcal{F}_0$ and $\ker(\mathcal{F}_0)_*$ is the kernel (i.e., 'nullspace') of $(\mathcal{F}_0)_*$, so that:

$$\ker(\mathcal{F}_0)_* = \left\{ \frac{\partial}{\partial u_4^*} \right\}. \tag{A.9}$$

Since it now can be calculated that $[\partial/\partial u_i^*, \ker(\mathcal{F}_0)_*] \subset \ker(\mathcal{F}_0)_*$ for $1 \leq i \leq 4$, the third condition is satisfied, and it can thus be concluded that the system (A.1) is feedback linearizable by the nonlinear feedback (A.5).

A.2 Lyapunov function's time-derivative

In order to find the time-derivative of the Lyapunov function that was introduced in Chapter 6:

$$V(t) = \frac{4}{6\mu m} \int_0^1 \left((m + \rho(x,t))^3 - (m + \rho_{ss})^3 \right)^2 dx, \tag{6.1}$$

an approach is used here that makes use of an entropy E and an entropic flux F [Cor02]. An entropy is a function $E : \rho \mapsto E(\rho)$, for which there exist a function $F : \rho \mapsto F(\rho)$, such that:

$$E_t + F_x = 0. \tag{A.10}$$

If the entropy function is now chosen in such a way that:

$$V(t) = \int_0^1 E(\rho) dx, \tag{A.11}$$

then the time-derivative of the Lyapunov function becomes:

$$\dot{V}(t) = -\left[F(\rho) \right]_{x=0}^{x=1}. \tag{A.12}$$

If the entropy function is defined as:

$$E(\rho) = \frac{4}{6\mu m} \left((m + \rho(x,t))^3 - (m + \rho_{ss})^3 \right)^2,$$

the entropic flux can be determined using (A.10) and (3.24):

$$\begin{aligned} F_x &= -E_t, \\ &= -E_\rho \cdot \rho_t, \\ &= -\frac{4}{6\mu m} \cdot 2 \left((m+\rho)^3 - (m+\rho_{ss})^3 \right) \cdot 3(m+\rho)^2 \cdot -\frac{\mu m}{(m+\rho)^2} \rho_x, \\ &= 4 \left((m+\rho)^3 - (m+\rho_{ss})^3 \right) \rho_x, \end{aligned} \tag{A.13}$$

$$F = (m+\rho)^4 - 4(m+\rho_{ss})^3\rho, \tag{A.14}$$

so that the time-derivative of the Lyapunov function becomes:

$$\begin{aligned} \dot{V} &= -\left[(m+\rho)^4 - 4(m+\rho_{ss})^3\rho\right]_{x=0}^{x=1}, \\ &= -(m+\rho_1)^4 + (m+\rho_0)^4 + 4(m+\rho_{ss})^3(\rho_1-\rho_0). \end{aligned} \tag{A.15}$$

Appendix B

Solving PDE-models analytically

Several analytical methods exist for solving first order PDEs, for example: the method of characteristics, which is described extensively in [McO03, Zac86, Zau89]. Another technique is the construction of integral surfaces of vector fields [Zac86] or the similar method of Lagrange [McO03]. The Cauchy-Kovalevsky theorem [Zac86] can be used to find an analytical solution (if it exists). For *systems* of first order PDEs, again the method of characteristics can be used. For some systems, the first order PDEs can also be solved separately. For example, consider Model 2 (3.20) and disregard the boundary conditions for the moment. The velocity function can now be solved separately from the density. Once solved, the velocity function is substituted in the mass conservation law and the density function can be solved. Not all systems, however, can be solved in this way. For example, if all equations in a system depend on more than one dependent variable, they can not be solved separately. When taking into account the boundary conditions in Model 2, the velocity is also dependent on the density (through the WIP) and thus it can not be solved separately from the mass conservation law. In this appendix, the attention only goes to the method of characteristics. The theory of this method is used to explain the concept of solution breaking.

B.1 Method of characteristics

Consider a first order PDE in one dependent variable, that is a function of place x and time t:

$$z_t + a(x,t,z)z_x = 0. \tag{B.1}$$

It is assumed that the solution $z(x,t)$ of this PDE can be found. In a 3-dimensional graph with on the axes x, t and $z(x,t)$, the solution forms a surface: for each position in the (x,t)-plane there is a corresponding value for z (see Figure B.1).

If, besides the PDE, the initial condition $z(x, t{=}0)$ of a system is known, the method of characteristics can be used to determine at least the solution in the neighbourhood

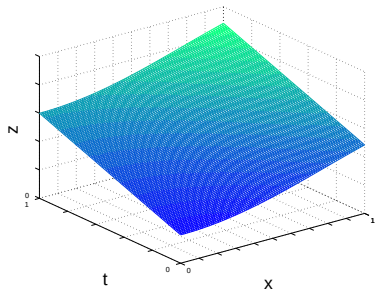


Figure B.1: A solution surface

of the initial condition. The method chooses a certain path (a *characteristic*) through the (x,t)-plane and determines the change of z along this path. Starting in an arbitrary point (x_0,t_0,z_0) on the initial curve, the path through the (x,t)-plane is determined by the derivative dx/dt which is defined by:

$$\frac{dx}{dt} = a, \tag{B.2}$$

in which a is equal to the coefficient in (B.1). Note that the derivative in (B.2) is a total derivative rather than a partial one. Using successively (B.2) and (B.1), the change of z along the path can be obtained:

$$\begin{aligned} \frac{dz(x,t)}{dt} &= \frac{\partial z}{\partial t} + \frac{\partial z}{\partial x}\frac{dx}{dt} \\ &= \frac{\partial z}{\partial t} + a\frac{\partial z}{\partial x} \\ &= 0. \end{aligned} \tag{B.3}$$

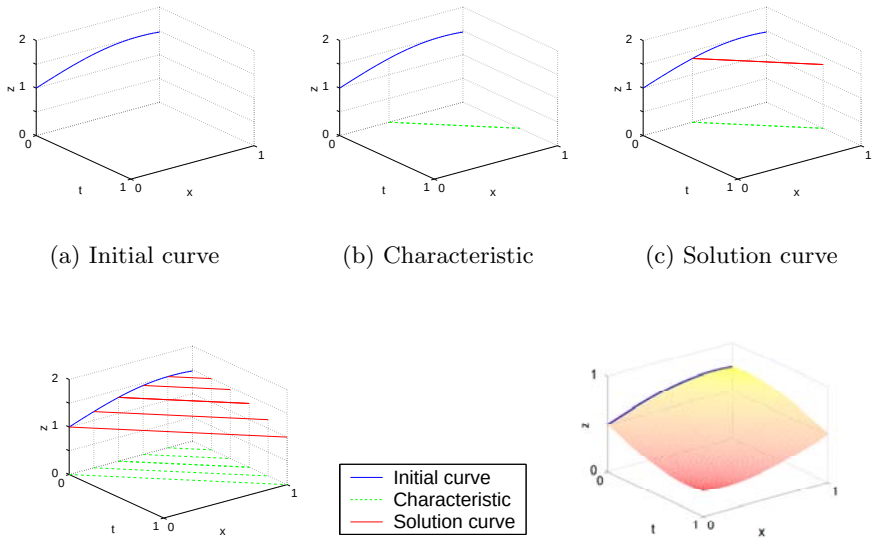
It can be seen that for (B.1) the solution curve z does not change along a path with derivative (B.2). Therefore, it can be concluded that the solution curve is only dependent on the starting point at the initial curve, time and coefficient a. Denoting the initial curve as $z(x, t{=}0) = f(x, t{=}0)$, the solution curve can be written in the form:

$$z(x_0, t) = f(x_0 - at). \tag{B.4}$$

Now, other characteristics can be drawn in a similar way, but with a different starting point on the initial curve. If an infinite number of characteristics are drawn, the neighbourhood of the initial curve will literally be covered with solution curves; together these curves form the solution surface. Similar to (B.4), the solution surface can now be written as:

$$z(x, t) = f(x - at). \tag{B.5}$$

Note that (B.4) and (B.5) are implicit functions (a is dependent on z), which makes the solution harder to interpret. However, this does not mean that (B.4) and (B.5) are



(a) Initial curve (b) Characteristic (c) Solution curve

(d) More solution curves (e) Solution surface

Figure B.2: Method of characteristics

useless. Valuable information on the solution can be derived from these equations, as is clarified in Appendix B.2.

The process of constructing characteristics and finding the solution surface is visualized in Figure B.2 for the situation in which $a = 1$. Here, either the initial values are assumed to be known for $x < 0$ or a boundary condition is assumed to be given at $x = 0$.

For the PDE in Figure B.2, a unique solution can be found for all $t > 0$. Unfortunately, this is not the case for all PDEs. Two problems can occur:

- If the initial curve is tangent to the characteristics at some place, the solution is not unique. Either no solution or infinitely many solutions can be found.
- If two characteristics cross each other, the solution at this intersection can have two values, and thus the solution is not unique anymore. In this case, the solution is only valid until the time the characteristics cross.

The first item indicates that to obtain a unique solution surface, the initial curve must be noncharacteristic: it may not be tangent to a characteristic at any place. The second item has to do with the development of shocks, which is discussed in Appendix B.2.

B.2 Solution breakage

In [Zau89] the concept of solution breakage is explained considering the first order PDE:

$$z_t + zz_x = 0, \tag{B.6}$$

in which $z(x,t) > 0$ under all circumstances. Here, the solution $z(x,t)$ is assumed to represent a wave in the ocean that moves to the right (direction of increasing x). The wave form at some time t^* is given by the curve $z(x,t^*)$ in the (x,z)-plane. The value of $z(x,t)$ represents the height of the wave at point x and time t. The characteristic of the wave is given by:

$$\frac{dx}{dt} = z, \tag{B.7}$$

which implies that the higher the wave, the greater the speed is of the corresponding point of the wave. Note that this is in contrast with (B.2), where the velocity is constant for all points. If initially there are higher parts of the wave located to the left of lower parts, the higher parts eventually catch up with and pass the lower parts. At this time the wave is said to break, and the solution $z(x,t)$ becomes multi-valued. The solution is therefore no longer valid.

In the description of physical processes with PDEs, $z(x,t)$ often represents a physical quantity such as density, which must be single valued under all circumstances. At the moment the solution breaks down and becomes multi-valued, the PDE describing the process is no longer an acceptable model for the physical process. In general, this means that some higher order derivative terms that were neglected in the construction of the PDE-model now become significant and cause the model to fail.

There are two solutions to the problem of breaking. The first solution is to extend the model, so that it will contain the significant higher order terms. This option is not discussed further here, since it falls beyond the scope of this project. The second solution to the breaking problem is to introduce a discontinuous solution known as the *shock wave*, which extends the validity of the solution beyond the breaking time. In this way, a solution for all $t > 0$ can be found based on the original first order PDE only.

Until the time of breaking, the solution found by the method of characteristics is valid. The interest here only goes to this part of the solution. For the construction of a solution beyond the breaking time the reader is referred to [Zau89]. For the calculation of the solution before the breaking time, it is also important to determine the breaking time. For a PDE of the form $z_t + a(z)z_x = 0$, this can be done by finding the time for which

$\partial z/\partial x$ becomes undefined. Using (B.5), this condition can be examined further:

$$\begin{aligned}\frac{\partial z}{\partial x} &= \frac{\partial f[x-a(z)t]}{\partial x} \\ &= \frac{df[x-a(z)t]}{d[x-a(z)t]} \cdot \frac{\partial [x-a(z)t]}{\partial x} \\ &= f'[x-a(z)t] \cdot [1 - t \cdot \frac{\partial a(z)}{\partial x}] \\ &= f'[x-a(z)t] \cdot [1 - t \cdot \frac{da(z)}{dz} \cdot \frac{\partial z}{\partial x}] \\ &= f'[x-a(z)t] - a'(z) \cdot t \cdot f'[x-a(z)t] \cdot \frac{\partial z}{\partial x},\end{aligned}$$

which can be solved for $\partial z/\partial x$:

$$\frac{\partial z}{\partial x} = \frac{f'[x-a(z)t]}{1 + a'(z) \cdot t \cdot f'[x-a(z)t]}. \tag{B.8}$$

From (B.8) it can be concluded that $\partial z/\partial x$ becomes undefined for:

$$1 + a'(z) \cdot t \cdot f'[x-a(z)t] = 0. \tag{B.9}$$

The moment of breaking can now be determined from (B.9) by solving it for t.

Besides finding a solution for all $t > 0$ in case breaking occurs, it is also interesting to find the cause of breaking and predict under which circumstances it will or will not occur. To avoid breaking of the solution, the left hand side of (B.9) must be either always positive, or always negative. The latter is disregarded, since for $t = 0$ the left hand side of (B.9) always equals 1. Keeping this in mind, a condition can be formulated that assures no breaking in the solution:

$$a'(z) \cdot f'[x-a(z)t] > -\frac{1}{t} \qquad \forall x, t \geq 0 \tag{B.10}$$

Since t is always positive and increasing, the right hand side of (B.10) asymptotically approaches zero. Therefore, it can be concluded that if $a'(z)f'[x-a(z)t]$ is always larger than or equal to zero, no breaking occurs. So, if $a'(z)$ is positive for all z, then f should be a non-decreasing function for all $t \geq 0$. On the other hand, if $a'(z)$ is negative for all z, then f should be non-increasing for all $t \geq 0$.

Consider the candidate PDE-models as presented in Section 3.3. Solution breaking is undesirable due to the discontinuous solution that it introduces. For Model 1, the velocity is independent of the place and therefore $a'(z) = 0$, which results in a solution without breaking, according to (B.10). Candidate Model 2 is harder to interpret, since it consists of two equations. No theoretical proof was found to state if (and under which circumstances) this model can result in a discontinuity in the solution. For Model 3, $a'(z) < 0$ for all z, so in order to avoid breaking the initial density function f should

be non-increasing for all $t \geq 0$. This implies that with this model only steady state and ramp-up situations can be modelled without causing a discontinuity in the solution. In Appendix C, however, it is explained that this PDE-model is solvable for ramp down situations as well, when using a numerical solver.

For further reading on solution breakage or shock waves the reader is referred to [Lax72, McO03, Zau89].

Appendix C

Solving PDE-models numerically

Besides analytically, PDEs can also be solved numerically [Hea97]. Several techniques are available of which the most relevant are mentioned here. A semidiscrete method, called the method of lines, discretizes the PDE with respect to the place and leaves the time continuous, so that it results in a system of ordinary differential equations (ODEs), for which many solution methods are known. Another semidiscrete method for PDEs approximates the solution by a linear combination of basis functions (functions over the spatial domain) with time-dependent coefficients. Finally, the fully discrete method discretizes the PDE with respect to both time and place.

In this appendix, the theory behind the fully discrete method is explored. During this exploration the concept of artificial diffusion is made clear as well.

C.1 Fully discrete method

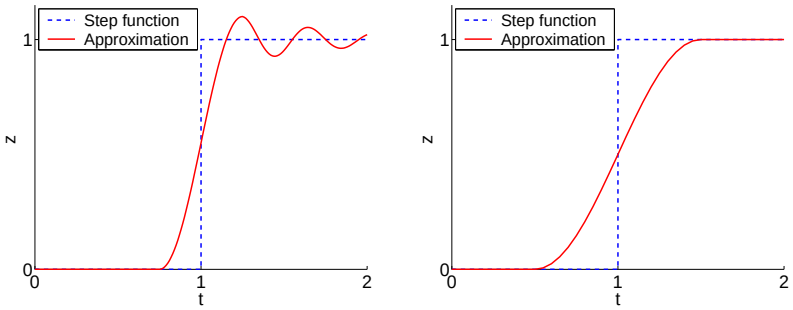
As mentioned before, a fully discrete method discretizes with respect to both time and place. This implies that all partial derivatives are replaced by finite difference approximations, for example:

$$z_x = \frac{z(x+dx,t) - z(x-dx,t)}{2dx}, \tag{C.1}$$

which is called *central spatial differencing*. Another differencing technique is called *upwind spatial differencing*:

$$z_x = \frac{z(x,t) - z(x-dx,t)}{dx}, \tag{C.2}$$

since it only uses only one side of the solution z for calculating z_x. Although the centered differencing formula is more accurate than the upwind differencing formula, the latter is suggested for solving hyperbolic PDEs. This is clarified with an example from [Hea97].



(a) Centered differencing (b) Upwind differencing

Figure C.1: Comparison of differencing methods for the approximation to a propagating step function of a linear PDE

Example C.1 (Centered versus upwind differencing) *Consider the linear PDE:*

$$z_t + z_x = 0,$$

with initial function z_0 *defined by:*

$$z_0(x) = \begin{cases} 1 & \text{if } x \leq 0, \\ 0 & \text{if } x > 0. \end{cases}$$

This discontinuity at $x = 0$ *in* z_0 *moves to the right with time. The value of each point in* $x \in \mathbb{R}^+$ *is 0, until the step function passes by, after which the value instantaneously becomes 1. This discontinuity causes a problem for differencing methods. In Figure C.1 the approximations of both differencing methods are visualized for position* $x = 1$.

Although the centered difference formula gives a better approximation of the sharp front, it also produces an overshoot and goes into an oscillation that is not present in the true solution. The upwind difference formula is less accurate in the approximation of the sharp front, but does stabilize at the true solution after a while. In many situations it is therefore a better solution.

Since an oscillating solution is undesirable for a PDE that describes a system which moves from one steady state to another, the upwind differencing technique is used to solve all candidate PDE-models.

Now, consider the main equations of Model 2 (3.19) as presented in Section 3.3:

$$\begin{aligned} \rho_t + (\rho v)_x &= 0, \\ v_t + v v_x &= 0. \end{aligned}$$

Replacing the partial derivatives by the finite difference approximations gives the following equations:

$$\begin{aligned} \frac{\rho(x,t+dt)-\rho(x,t)}{dt} + \frac{\rho(x,t)v(x,t)-\rho(x-dx,t)v(x-dx,t)}{dx} &= 0, \\ \frac{v(x,t+dt)-v(x,t)}{dt} + \frac{1}{2}\frac{v^2(x,t)-v^2(x-dx,t)}{dx} &= 0. \end{aligned} \tag{C.3}$$

Note that before replacing the term vv_x with a finite difference approximation, it was first transformed:

$$v\frac{\partial v}{\partial x} = \frac{1}{2}\cdot 2v\cdot\frac{\partial v}{\partial x} = \frac{1}{2}\frac{dv^2}{dv}\frac{\partial v}{\partial x} = \frac{1}{2}\frac{\partial v^2}{\partial x}. \tag{C.4}$$

Now (C.3) can be rearranged and transformed to obtain:

$$\begin{aligned} \rho(x_i,t_{j+1}) &= \rho(x_i,t_j) - \frac{dt}{dx}[\rho(x_i,t_j)v(x_i,t_j) - \rho(x_{i-1},t_j)v(x_{i-1},t_j)] \\ v(x_i,t_{j+1}) &= v(x_i,t_j) - \frac{1}{2}\frac{dt}{dx}[v^2(x_i,t_j) - v^2(x_{i-1},t_j)]. \end{aligned} \tag{C.5}$$

Here x_i and t_j denote respectively sample i in place and sample j in time. For calculation of the density and velocity at the first sample in place, $\rho(x_{i-1},t)$ and $v(x_{i-1},t)$ are equal to the boundary conditions at time t. Starting with the initial state and the boundary conditions for the next sample in time, the values of all samples in x at the next sample in time can be calculated. Subsequently, this procedure can be repeated for each new sample in time. For Models 1 and 3, the procedure of approximation and solving is identical and is therefore not worked out here.

For the fully discrete method the accuracy of the solution depends on the step sizes in both place and time. Note that the accuracy does not necessarily increase with decreasing step sizes. The step sizes can not be chosen independently of each other. To converge to the true solution of the PDE as step sizes approach zero, two conditions must be met:

- *Consistency*: the local truncation error must go to zero as the step sizes go to zero, which implies that the *correct* continuous problem is approximated.
- *Stability*: the approximation solution must remain bounded.

According to Lax Equivalence Theorem [Hea97] these conditions are necessary and sufficient to guarantee convergence. For the approximation of the candidate PDE-models the first condition is met. The second condition can be translated into the following condition:

$$dt \leq \frac{dx}{v_{\max}(t)}, \tag{C.6}$$

in which $v_{\max}(t)$ is the maximum of all occurring velocities in the system at time t. Since this condition is too strict, the approximation of the PDE-models is subject to artificial diffusion, as discussed in Appendix C.2.

C.2 Diffusion

Here, diffusion is defined as smoothing sharp fronts in the solution or as mixing of values at adjacent samples. A good example of diffusion can be seen in Figure C.1(b), in which the sharp front of the step function is approximated with a smoother numerical solution. The kind of diffusion described in this figure is called artificial diffusion, since it is added to the PDE-model as a result of the numerical approximation (the true solution of the PDE-model is the step function).

The questions that now arise are: what causes this artificial diffusion and is it desirable? To start with the first question, the diffusion is the result of the stability condition (C.6), which is too strict. In (C.6) the term $v_{\max}$ appears. This means that the step size in time dt is dependent on the maximum velocity at that moment. If the velocity is not uniform in the entire system, then dt is smaller than necessary for the places in the system where the velocity is lower. This phenomenon — that dt is smaller than necessary for stability — causes artificial diffusion. This will be explained further by means of Model 3 (with $m = 1$), as presented in Section 3.3. Assume for the moment that, in each sample dx, dt can be adjusted to the local velocity v, so that locally:

$$\frac{dt}{dx} = \frac{1}{v} = \frac{1+\rho}{\mu}. \tag{C.7}$$

Substituting (C.7) into the finite difference approximation for Model 3 gives:

$$\begin{aligned} \rho(x_i, t_{j+1}) &= \rho(x_i, t_j) - \frac{dt}{dx}\left[\frac{\mu\rho(x_i, t_j)}{1+\rho(x_i, t_j)} - \frac{\mu\rho(x_{i-1}, t_j)}{1+\rho(x_{i-1}, t_j)}\right] \\ &= \rho(x_i, t_j) - \rho(x_i, t_j) + \frac{1+\rho(x_i, t_j)}{1+\rho(x_{i-1}, t_j)}\rho(x_{i-1}, t_j) \end{aligned} \tag{C.8}$$

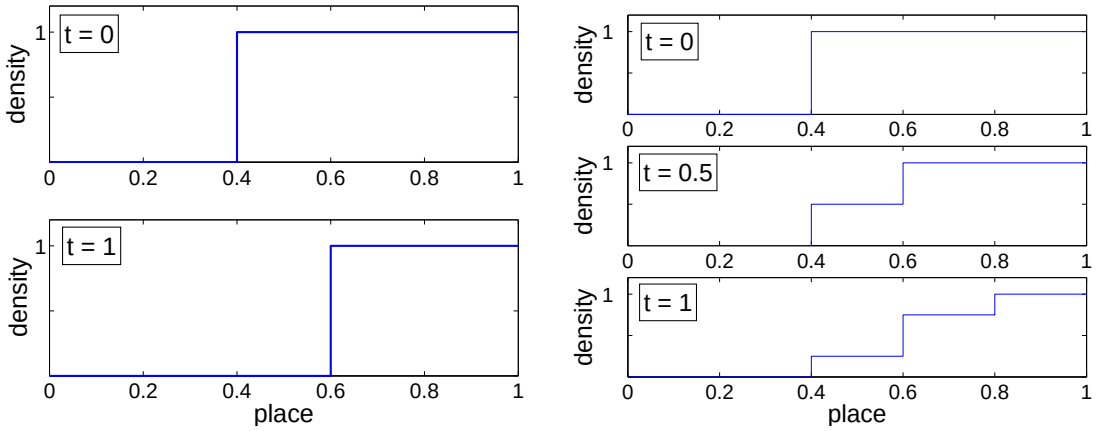
Obviously, the old density at x_i is replaced by a term that is almost entirely dependent on the density at x_{i-1}.

Now, consider what happens if dt is smaller than necessary for stability. Here ρ^* is the density that is coupled to the maximum velocity with which dt is determined. Note that ρ^* is smaller than $\rho(x_i, t_j)$ and $\rho(x_{i-1}, t_j)$.

$$\frac{dt}{dx} = \frac{1}{v_{\max}} = \frac{1+\rho^*}{\mu}, \tag{C.9}$$

$$\rho(x_i, t_{j+1}) = \rho(x_i, t_j) - \frac{1+\rho^*}{1+\rho(x_i, t_j)}\rho(x_i, t_j) + \frac{1+\rho^*}{1+\rho(x_{i-1}, t_j)}\rho(x_{i-1}, t_j). \tag{C.10}$$

Here the new density at x_i clearly is a mix of the old densities at x_i and x_{i-1}. Since dt is smaller ($v_{\max} \geq v$), the effect of the old density at x_{i-1} is now sooner penetrated in the new density of x_i. This is artificial diffusion, as is visualized in Figure C.2.



(a) No artificial diffusion ($v = v_{\max}$)

(b) Artificial diffusion ($v < v_{\max}$)

Figure C.2: The effect of artificial diffusion

From the above reasoning it can also be concluded that artificial diffusion is not desirable, since it changes the behaviour of the original PDE-model. For example, consider a controller that is based on a certain PDE-model. If this controller is applied in a simulation, the PDE-model (solved with the fully discrete method) describes different behaviour, which affects the performance of the controller.

On the other hand, if not the original PDE-model but the numerically approximated PDE-model (with artificial diffusion) is considered for the design of a controller, several advantages of artificial diffusion can be identified.

First of all, regarding a flow line, several phenomena can be found that suggest diffusion. For example, in case of a ramp down, the mean buffer content of the second machine in line starts decreasing right after the mean buffer content of the first machine decreased. If no diffusion would have been present, the first buffer would empty all the way to the new mean content before the second buffer would start to empty. For PDE-models that originally do not describe (sufficient) diffusion, artificial diffusion might be a solution, especially since the amount of artificial diffusion can be influenced (to a certain extent).

Secondly, the artificial diffusion is one-way diffusion: products do not flow backwards (upstream). Generally, if diffusion explicitly appears in a PDE-model, it is a second order term. As mentioned before, Daganzo [Dag95] disapproved the second order traffic models, since the diffusion term worked in two directions: under certain circumstances cars would drive backwards on the highway.

Now consider the candidate models as presented in Section 3.3. For Model 1, the numerical approximation method does not add artificial diffusion, since the velocity is uniform throughout the whole system. Therefore, the local velocity at each point in the line equals the maximum velocity, and no artificial diffusion is introduced. For Models 2

and 3, on the other hand, the numerical approximation method adds artificial diffusion.

In Appendix B.2 it was mentioned that Model 3 could not describe ramp down situations. However, with the introduction of (enough) artificial diffusion the breaking of the solution does not occur, and thus ramp down can be described, as can be seen in Chapter 4.

Appendix D

Validation model

D.1 χ code

The example of the χ code presented here, is the code for simulation of a $G(BM)^{10}E$-line (i.e., an M/M/1 line with 10 identical machines), which is subject to a ramp down. In Appendix D.2, the code of this file is explained and it is described how adjustments to this file can be made in order to simulate other situations, such as ramp up, a different number of machines, another target utilization or a different initial state.

```
// m10u50cs1d.chi RAMPDOWN
// A 'G (BM)^10 E' line with exponential arrivals and process times is simulated.
// Homogenous processes with a utilization of 50% are reduced to a utilization of 25%.
// CT, TT, and WIP per process are determined and written to a file each time a lot
// arrives at the exit process.

type lot = real#nat

proc G(a: !lot, l: real) =
|[ i: nat, td: ->real
 | i:= 1
 ; td:= negexp(l)
 ; *[ delta sample td; a!<time,i>; i:= i + 1]
]|

proc B(a: ?lot, b: !lot, c: !nat, d,mach: ?void, bufi: nat) =
|[ x: lot, xs: lot*, send: bool, m: nat
 | xs:= []; send:= false; m:= 0
 ; *[ len(xs) < bufi -> xs:= xs ++ [<0.0,0>] ]
 ; *[ true       ; a?x             -> xs:= xs ++ [x]
    | len(xs) > 0; b!hd(xs)        -> xs:= tl(xs)
                                    ; m:= 1
    | not send   ; d?              -> send:= true
    | send       ; c!(len(xs) + m) -> send:= false
    | true       ; mach?           -> m:= 0
    ]
]|

proc M(a: ?lot, b: !lot, mach: !void, l:real) =
```

```
|[ x: lot, td: ->real
 | td:= negexp(l)
 ; *[ a?x; delta sample td; b!x; mach!]
]|

proc E(a: ?lot, c: (?nat)^10, d: (!void)^10) =
|[ x: lot, CT,TT,toud: real, buf,i: nat
 | toud:= 0.0; i:= 0; CT:= 1.0
 ; *[ a?x
    ; [ x.0 > 0 -> CT:= time-x.0
      | x.0 = 0 -> skip
      ]
    ; TT:= time-toud
    ; toud:= time
    ; !time, tab(), CT, tab(), TT, tab()
    ; *[ i < 10 -> d.i!; c.i?buf
                  ; ! tab(), buf
                  ; i:= i + 1
       ]
    ; ! nl()
    ; i:= 0
    ]
]|


syst S(l1,l2: real, bufi: nat) =
|[ a: (-lot)^11, b: (-lot)^10, c: (-nat)^10, d,e: (-void)^10
 | G(a.0,l1)
|| i: nat <- 0..10: B(a.i,b.i,c.i,d.i,e.i,bufi)
|| i: nat <- 0..10: M(b.i,a.(i+1),e.i,l2)
|| E(a.10, c, d)
]|

xper = |[ S( 1/0.5 // interarrival time (hours/lot)
           , 1/2.0 // process time (hours/lot)
           , 1     // initial buffer content (for all processes)
           ) ]|
```

D.2 Model description

The validation model presented in Appendix D.1 is composed of 4 different processes and one system, which are described here one by one. After this global description, it is discussed how the file can be adapted for simulation of other situations.

Process G is the generator process, which simulates the arrival of lots. The time between the arrival of two subsequent lots is exponentially distributed, with a mean value that is defined in the *xper*-environment. After this time has elapsed a lot is sent to the buffer of the first process. The information that is coupled to the lot is: the time that it entered the system and an identification number.

Process B represents a buffer process. The process receives lots from a previous process (either the generator process or a machine process), stores them in a list, and sends them on to the next machine process whenever this process is ready to receive a lot. Of course, in order to send a lot to the next machine, at least one lot must be present

in the list. Furthermore, the process keeps up the number of lots (WIP) present in the process area, that is, in the buffer and the corresponding machine. When a lot is sent to the machine, the number of lots in the buffer decreases with one (the length of the list becomes one lot shorter) and the number of lots in the machine becomes one. When the machine has finished processing a lot and has sent it on, it sends a signal to the buffer to notify that the machine is empty. Each time a lot has reached the exit process, the WIP-level for all process areas is printed to a file. In case of a ramp down simulation, the initial buffer contents are not empty; the number of products in the buffer is defined in the *xper*-environment. In this case, the waiting list is filled with the correct number of lots, before the actual simulation starts. All these lots get identification number 0 and starting time 0.0, so that they can easily be distinguished in the exit process.

Process M represents a machine process, which describes the time a lot spends in a machine. Once a lot is received from the corresponding buffer process, it is delayed with a certain time, that represents the process time. This time is exponentially distributed and the mean is defined in the *xper*-environment. After the time has elapsed, the lot is sent on to the next buffer process (or the exit process, in case of the last machine) and a signal is sent to the buffer in front of the machine to notify that the machine is empty. Then, the process is ready to receive a lot again.

Process E is the exit process, at which lots arrive once they have been processed at all machines. When a lot arrives at this process, the flow time (here: CT) and inter departure time (here: TT) are determined, the WIP-levels in all process areas are collected and all these data are written to a file. (In fact the data are written to the standard output, which by the .sh file that is described in Appendix D.3, is redirected to a file.) In case of a ramp down simulation, the lots that were initially in the system, do not possess information from which the flow time can be determined. Therefore, the flow time is set to 1 for all these lots (setting the value to zero would cause problems in processing the output data). The other output data (TT and WIP) are determined in the way described above.

System S describes the structure of the system that is simulated, using the defined processes. In the χ code in Appendix D.1, the system consists of one generator process followed by 10 machine processes with corresponding buffer processes and closed by the exit process.

Adapting the χ code

The χ code can be adapted in various ways, in order to simulate other situations. Here, the changes required for ramp up, i.e., a different number of machines in the system, a different target utilization, and a different initial state, are discussed.

For the ramp up situations as described in this research, the initial buffer content is always zero. This value can be changed in the *xper*-environment.

The number of machines in the system can be changed by adapting the values in process E and system S. In process E, the number of channels in the bundle should be changed (both in the declaration of ports and in the repetitive selective waiting statements). In system S, the number of channels in the bundles should be changed, the number of buffer- and machines processes should be adapted, and the input channel of the exit process should be modified. Note that the number of channels in bundle a is one larger than the number of machines in the system.

The target utilization can be altered by modifying the arrival rate, which is defined in the *xper*-environment. The description of the initial utilization can be realized by means of the initial buffer content.

The initial state is defined by the number of lots in the buffer. As mentioned before, this value can be changed in the *xper*-environment. Note that only initial states are considered in which the system is in steady state, so that the buffer content is identical for each process.

D.3 Running 100 similar simulations

For the validation method is described in Appendix F it is necessary that a set of 100 simulations is run with one χ-model. In order to arrange these simulations and their output files in an orderly way, a *Bourne shell* is used. This file, with the same name as the χ-file, but with an .sh extension, has the following content:

```
./m10u50cs1d -e 6000 > expDE1.txt
./m10u50cs1d -e 6000 > expDE2.txt
./m10u50cs1d -e 6000 > expDE3.txt

           ...

./m10u50cs1d -e 6000 > expDE98.txt
./m10u50cs1d -e 6000 > expDE99.txt
./m10u50cs1d -e 6000 > expDE100.txt
```

The Bourne shell executes the simulations one after another and writes all output data of one simulation to a separate file (expDE1.txt – expDE100.txt). A simulation is started with the command *./m10u50cs1d* and is terminated once the defined simulation time is reached (here: 6000). It is noted that for simulation of other situations, this simulation time should be changed as well. For example: simulations with more machines need a longer simulation time before steady state is reached.

Appendix E

PDE-models

In this research, the fully discrete method has been used to solve the candidate PDE-models. For this method an algorithm was already available in a C^{++} model [Jo03]. The model has been adapted for this investigation and is described in this appendix. Since the implementation of the fully discrete method for the candidate PDE-models is in fact the numerical approximation of the PDE-models, the implementation in C^{++} is here also simply referred to as 'PDE-model'.

E.1 General description of the PDE-model

The PDE-model is written in C^{++} and consists of various files. These files and their functions in the PDE-model are sorted and arranged in the tables below.

Main file	
onefac.cpp	(see Appendix E.3)

Input files	
fact0001.in	Definition of process rate μ, number of machines m and number of steps in x-direction nx.
influx.cpp	Definition of influx as function of time.
initM10_d.cpp	Definition of initial density as function of place.
initM10_mu.cpp	Definition of process rate as function of place (only necessary if this is not a constant value; only necessary for Model 3)
initM10_v.cpp	Definition of initial velocity as function of place (only necessary for Model 2).
model.in	Choice of the candidate model that is used for the simulation

Output generating files	
scalarout.cpp	Collect necessary output data and generate *scalar.data*.
solnout.cpp	Collect necessary output data and generate *soln****.data*.

Output files	
scalar.data	Contains general output data. The columns represent respectively: time, influx, WIP, mean velocity, and outlux.
*soln****.data*	Time depending output data. The columns represent respectively: time, place, density, velocity, flux.

Model files	
exactmod.cpp	Candidate Model 2, $v(\rho)$ relation according to queueing theory.
exactmods.cpp	Candidate Model 1, $v(\rho)$ relation according to queueing theory.
expmod.cpp	Candidate Model 2, $v(\rho)$ relation approximated with exponential function.
expmods.cpp	Candidate Model 1, $v(\rho)$ relation approximated with exponential function.
exppar.cpp	Determines parameters for *expmod.cpp* and *expmod.cpp*.
linmod.cpp	Candidate Model 2, $v(\rho)$ relation approximated with linear function.
linmods.cpp	Candidate Model 1, $v(\rho)$ relation approximated with linear function.
linpar.cpp	Determines parameters for *linmod.cpp* and *linmod.cpp*.
mm1mod.cpp	Candidate Model 3
quadmod.cpp	Candidate Model 2, $v(\rho)$ relation approximated with quadratic function.
quadmods.cpp	Candidate Model 1, $v(\rho)$ relation approximated with quadratic function.
quadpar.cpp	Determines parameters for *quadmod.cpp* and *quadmod.cpp*.

Other files	
basicmath.cpp	Definition of mathematical functions.
factory.h	Definition of the class 'factory'.
factory.cpp	Storage of the parameters of the class 'factory'.
fnamech.cpp	Definition of filenames(?) The purpose of this file is not clear.
integrleft.cpp	Definition of numerical integration.
intel.h	Definition of files in the PDE-model.
makefile	Compilation and simulation of the PDE-model.

In a Linux environment, a simulation can be started by the command: *make runonefac*.

Adapting the C++ code

The C++ code can be adapted in various ways, in order to simulate other situations. Here, the changes needed for simulation with a different candidate model, a different number of machines in the system, a different target utilization, and a different initial state are discussed. Furthermore, changing the end time of a simulation, *nx*, *cflcond* or *nsave* will be discussed as well.

The choice of the candidate model can be made in the input file *model.in*. The number of machines can be changed in input file *fact0001.in*. The target utilization can be changed by modifying the arrival rate, which is defined in *influx.cpp*. The initial density can be adapted in *initM10_d.cpp* and for Model 2 the initial velocity can be changed in *initM10_v.cpp*. For the case in which the initial state is a steady state, which is the case in all performed experiments, the initial velocity is calculated automatically, and thus no change is necessary.

In *fact0001.in* the number of steps in x-direction can be changed. The main file contains the definitions of the simulation end time, *cflcond* and *nsave*.

E.2 C++ code

Since the model is very large and divided over many files, the entire C++ code will not be given here. Instead, only the C++ code of the main file *onefac.cpp* will be given. For the rest of the C++ code, the interested reader is referred to the CD-ROM that goes with this report. The C++ code presented below will be described in Appendix E.3.

```
//=========================================================
//
//                onefac.cpp (main program)
//
//=========================================================

#include <fstream.h>
#include <stdlib.h>
#include <math.h>
#include <string.h>
#include "basicmath.h"
#include "intel.h"
#include "factory.h"

int main(void){

  // ------------    Input 1 (begin)  -------------------
  //
  // !!!! --- Modify "fact0001.in" --- !!!
  // !!!! --- Modify "model.in"    --- !!!
  // !!!! --- Modify "influx.cpp"  --- !!!
  //
  // !!!! --- Modify initial conditions ---!!!
  //          (density, process rate, velocity)
  //
```

```
double endtime=300.0;   // end_time for simulation
int nsave=30;           // for saving output - to generate a soln****.data
                        //                     file every 'nsave' times
double cflcond=0.5;     // should be less than 1.
                        // (stability issue)

// ------------    Input 1  (end)   -------------------



int nfac=1;    // the number of factory (do not change)

char infile1[12]={'f','a','c','t','0','0','0','0','.','i','n','\0'};
char infile2[9]={'m','o','d','e','l','.','i','n','\0'};

char outfile1[13]={'s','c','a','l','a','r','.','d','a','t','a','\0'};
char outfile2[14]={'s','o','l','n','0','0','0','0','.','d','a','t','a','\0'};

int incnt=0;     // count for input file
int ninfo1=1;    // the number of input for a factory class (double)
int ninfo2=2;    // the number of input for a factory class (int)
int model[nfac][2];

double *fd=new double[ninfo1];  // mu
int *fi=new int[ninfo2];        // m, nx

int mod, mods;
double rawvel, alpha, beta;

int i,j,k;

factory * fact=new factory[nfac];

ifstream fin;

for (i=0;i<nfac;i++){
  incnt++;
  fnamech(incnt, infile1);

  fin.open(infile1);

  for (j=0;j<ninfo1;j++){
    fin >> fd[j];
  }
  for (k=0;k<ninfo2;k++){
    fin >> fi[k];
  }
  fin.close();

  fact[i]=factory(fd,fi);     // mu, m, nx are here DEFINED in factory.cpp
}

// ------------    Input 2 (begin)  -------------------
//
// Give data set for each factory.
// (Not perfect yet)
// (Only necessary when expmod(s), linmod(s) or quadmod(s) is used)
//

double data[nfac][4];
double data1[2], data2[2];
```

```
// Two data points of equilibrium, flow time (tpt) and Work In Process (wip)
data[0][0]=100.0;       // tpt    [hour]
data[0][1]=190;         // wip    [parts in whole factory]


data[0][2]=20.0;        // tpt    [hour]
data[0][3]=30;          // wip    [parts in whole factory]


// ------------    Input 2  (end)   -------------------


//################################################
//################################################
//
//               simulation begins
//
//################################################
//################################################

double time, tstep, cfl[nfac], dt[nfac];
double step[nfac], dx;
double wip[nfac], meanv[nfac];
double infx[nfac], outfx[nfac];
double scalar[5];
double maxvel[nfac], tempvel;

int nn, count, fcount=0;
int nx[nfac], mx, nxmax;
int m;


// ---- read 'infile2' ----
// determine model type

fin.open(infile2);
for (i=0;i<nfac;i++){
  for(j=0;j<2;j++){
    fin >> model[i][j];
  }
}
fin.close();

for (i=0;i<nfac;i++){
  mod=model[i][0];
  mods=model[i][1];
  fact[i].model_factory(mod, mods);      //mod, mods are WRITTEN to factory.cpp
}

// --- find proper 'alpha' and 'beta'
for (i=0;i<nfac;i++){
  fact[i].getm_factory(mod, mods);       //mod, mods are READ from factory.cpp
  fact[i].getp_factory(rawvel,alpha,beta); // rawvel, alpha, beta are READ from factory.cpp

  for(j=0;j<2;j++){
    data1[j]=data[i][j];
    data2[j]=data[i][j+2];
  }

  if (mod==1){
    cout << " ==== Factory M"<<i+1<<"0 ==== \n";
```

```
    cout << " Linear model \n";
    linpar(data1,data2,rawvel,alpha,beta);
  }
  else if (mod==2){
    cout << " ==== Factory M"<<i+1<<"0 ==== \n";
    cout << " Quadratic model \n";
    quadpar(data1,data2,rawvel,alpha,beta);
  }
  else if (mod==3){
    cout << " ==== Factory M"<<i+1<<"0 ==== \n";
    cout << " Exponential model \n";
    exppar(data1,data2,rawvel,alpha,beta);
  }
  else if (mod==4){
    cout << " ==== Factory M"<<i+1<<"0 ==== \n";
    cout << " 'Exact' model (based on Queuing Theory) \n";
  }
  else if (mod==5){
    cout << " ==== Factory M"<<i+1<<"0 ==== \n";
    cout << " M/M/1 model \n";
  }
  fact[i].parm_factory(rawvel,alpha,beta); //rawvel, alpha, beta are WRITTEN to factory.cpp
  fact[i].info_factory();            //prints factory information to the screen
}

nxmax=0;
for (i=0;i<nfac;i++){
  fact[i].getnx_factory(mx);
  nx[i]=mx;
  if (nxmax<mx){
    nxmax=mx;
  }
  step[i]=1.0/(mx-1);
}

double x[nfac][nxmax];
double defmu[nfac][nxmax];
double rho[nfac][nxmax];
double newrho[nfac][nxmax];
double vel[nfac][nxmax];
double newvel[nfac][nxmax];

time=0.0;
count=0;
fcount=0;

// ------  get x-values -----

for (i=0; i<nfac; i++){
  dx=step[i];
  for (j=0;j<mx; j++){
    x[i][j]=(double)(j)*dx;
  }
}

// -----   get process time values   -----

for (j=0; j<nx[0]; j++){
  defmu[0][j]=initM10_mu(x[0][j]);
}

// -----   get initial densities   -----
```

```
for (j=0; j<nx[0]; j++){
  rho[0][j]=initM10_d(x[0][j]);
}

// ----- get wip -----

for (i=0;i<nfac;i++){
  wip[i]=integrleft(rho[i], 0, nx[i]-1, 0, 1);
}


// ----- get initial velocity -----

for (i=0;i<nfac;i++){
  fact[i].getp_factory(rawvel,alpha,beta); //Rawvel, alpha, beta are READ from factory.cpp
  fact[i].getmach_factory(m);
  if (mod==5){   // for the case that the model is based on the M/M/1
    if(i==0){
      for (j=0;j<nx[i];j++){
        vel[i][j]=(defmu[i][j]/m)/(1+(rho[i][j]/m));
      }
    }
    meanv[i]=integrleft(vel[i], 0, nx[i]-1, 0, 1);
  }
  else {
    if (mods==0){  // for the case that velocity depends only on time
      if (mod==1){  // Linear model
    tempvel=alpha/(1.0+beta*wip[i]);
    for (j=0; j<nx[i]; j++){
      vel[i][j]=tempvel;
    }
      }
      else if (mod==2){  // Quadratic model
    tempvel=rawvel/(1.0+alpha*wip[i]+beta*wip[i]*wip[i]);
    for (j=0; j<nx[i]; j++){
      vel[i][j]=tempvel;
    }
      }
      else if (mod==3){ // Exponential model
    if (wip[i]<1.0e-8){
      tempvel=rawvel;
    }
    else {
      tempvel=alpha*(1.0-exp(-beta*wip[i]))/wip[i];
    }
    for (j=0; j<nx[i]; j++){
      vel[i][j]=tempvel;
    }
      }
      else if (mod==4){ // 'Exact' model (based on queuing theory)
        tempvel=alpha/(1.0+beta*wip[i]);
        for (j=0; j<nx[i]; j++){
          vel[i][j]=tempvel;
        }
      }
      meanv[i]=tempvel;
    }

    else {
      if(i==0){
    for (j=0;j<nx[i];j++){
```

```
          tempvel = alpha/(1.0+beta*wip[i]);
      vel[i][j]=initM10_v(tempvel, x[i][j]);
    }
      }

      meanv[i]=integrleft(vel[i], 0, nx[i]-1, 0, 1);
    }
  }
}

// ---- influx ----

infx[0]=influx(time);

// ---- outflux ----

for (i=0;i<nfac;i++){
  outfx[i]=rho[i][nx[i]-1]*vel[i][nx[i]-1];
}

scalar[0]=time;
for(i=0;i<nfac;i++){
  scalar[i*4+1]=infx[i];
  scalar[i*4+2]=wip[i];
  scalar[i*4+3]=meanv[i];
  scalar[i*4+4]=outfx[i];
  nn=i*4+5;
}
scalarout(outfile1, scalar, nn);
for(i=0;i<nfac;i++){
  solnout(outfile2, time, i, nx[i], rho[i], vel[i]);
}                          // time, x, density, velocity


// ***********************************************
// **********  time loop (begin)  ****************
// ***********************************************


while((time < endtime) && (count<100000) && (fcount < 5000)){

  // -----   get info for the next simulation   -----

  for (i=0;i<nfac;i++){
    maxvel[i]=0.0;
    if (model[i][1]==0){
  if (vel[i][0] > maxvel[i]){
    maxvel[i]=vel[i][0];
  }
    }
    else {
  for(j=0;j<nx[i];j++){
    if (vel[i][j] > maxvel[i]){
      maxvel[i]=vel[i][j];
    }
  }
    }
  }

  tstep=1.0;
  for(i=0;i<nfac;i++){
    dt[i]=cflcond*step[i]/maxvel[i];
```

```
  if (dt[i] < tstep){
tstep=dt[i];
  }
}

for (i=0;i<nfac;i++){
  cfl[i]=tstep/step[i];
}

infx[0]=influx(time);

// ----- get new velocity, density, wip, meanv -----

for(i=0;i<nfac;i++){
  fact[i].getp_factory(rawvel,alpha,beta);

  if (mod==5){
    mm1mod(cfl[i],wip[i],meanv[i],infx[i],nx[i],m,
       rho[i],vel[i],newrho[i],newvel[i],defmu[i]);
  }
  else {
    if (mods==0){
  if (mod==1){
    linmods(cfl[i],wip[i],meanv[i],infx[i],nx[i],rho[i],
        vel[i],newrho[i],newvel[i],alpha,beta);
  }
  if (mod==2){
    quadmods(cfl[i],wip[i],meanv[i],infx[i],nx[i],rawvel,
         rho[i],vel[i],newrho[i],newvel[i],alpha,beta);
  }
  if (mod==3){
    expmods(cfl[i],wip[i],meanv[i],infx[i],nx[i],rawvel,
        rho[i],vel[i],newrho[i],newvel[i],alpha,beta);
  }
  if (mod==4){
    exactmods(cfl[i],wip[i],meanv[i],infx[i],nx[i],rho[i],
          vel[i],newrho[i],newvel[i],alpha,beta);
  }
    }
    else {
  if (mod==1){
    linmod(cfl[i],wip[i],meanv[i],infx[i],nx[i],rho[i],
       vel[i],newrho[i],newvel[i],alpha,beta);
  }
  if (mod==2){
    quadmod(cfl[i],wip[i],meanv[i],infx[i],nx[i],rawvel,rho[i],
        vel[i],newrho[i],newvel[i],alpha,beta);
  }
  if (mod==3){
    expmod(cfl[i],wip[i],meanv[i],infx[i],nx[i],rawvel,
       rho[i],vel[i],newrho[i],newvel[i],alpha,beta);
  }
  if (mod==4){
    exactmod(cfl[i],wip[i],meanv[i],infx[i],nx[i],rho[i],
         vel[i],newrho[i],newvel[i],alpha,beta);
  }
    }
  }
}

time+=tstep;
count++;
```

```
    // ----- update new velocity and density -----

    for(i=0;i<nfac;i++){
      for (j=0;j<nx[i];j++){
    vel[i][j]=newvel[i][j];
    rho[i][j]=newrho[i][j];
      }
    }

    // ---- outflux ----

    for (i=0;i<nfac;i++){
      outfx[i]=rho[i][nx[i]-1]*vel[i][nx[i]-1];
    }


    scalar[0]=time;
    for(i=0;i<nfac;i++){
      scalar[i*4+1]=infx[i];
      scalar[i*4+2]=wip[i];
      scalar[i*4+3]=meanv[i];
      scalar[i*4+4]=outfx[i];
      nn=i*4+5;
    }

    scalarout(outfile1, scalar, nn);

    if ((count%nsave)==0){ // for saving output
      fcount++;
      fnamech(fcount,outfile2);
      for(i=0;i<nfac;i++){
    solnout(outfile2, time, i, nx[i], rho[i], vel[i]);
      }                          // time, x, density, velocity
    }
  }

 return(0);
}
```

E.3 Description of *onefac.cpp*

The file *onefac.cpp* is the main file of the PDE-model. Its functions can be split into three parts: gathering all the necessary input data, supervise the calculations in the simulation, and collect output data in an orderly way. These parts are now worked out.

In the first part of the main file the simulation end time, *nsave* and *cflcond* are defined, and other parameters are read from the input files (see Appendix E.1). The parameters that describe the behaviour of the manufacturing system and the candidate model are stored in the class 'factory'. At the end of the input part, there is a section under the name 'input 2'. In this section, which is only used if the linear, quadratic or exponential approximation model is applied, two data points are defined. These data points are used to obtain a linear, quadratic or exponential approximation of the flow time as function of the throughput, or actually: an approximation of the relation between density and

velocity. For the case in which the structure of the manufacturing system is known, this method of obtaining the $v(\rho)$ relation is not effective and might even give a wrong solution when using the wrong approximation. However, for systems with an unknown structure, this method might be useful. Of course, it could be improved, for example by increasing the number of data points and using the least squares method to obtain an approximation of the $v(\rho)$ relation.

In the second part, first the initial conditions are determined and the step size in time is determined. Then the necessary data are sent to one of the models (see Appendix E.1), in which the next state of the system is calculated. The new data are sent back to the main file, in which they are arranged for output and used for the preparation of the next iteration. The iterations go on until either the simulation end time, or the maximum number of iterations, or the maximum number of output files has been reached. The limits for the last two numbers are defined at the start of the first iteration (in *onefac.cpp* this point is indicated by 'time loop (begin)').

In the last part, all output data are collected and sent to the output generating files (see Appendix E.1), in which they are orderly written to output files. Note that this data collection does not only appear at the end of each iteration, but also before the first iteration starts; the initial state data are written to the output files as well.

Appendix F

Method of validation

The DEM and the PDE-models, as described in respectively Section 2.3 and Section 3.3, produce different kinds of output. In order to validate and compare the candidate models, the output of both the PDE-models and the validation model should be transformed into useful data, i.e., data from which the performance measures, as presented in Section 4.1, can be derived. In order to achieve this, the output data of both the PDE-models and the DEM should be processed. This processing is described here.

F.1 Output processing for the PDE-models

The output data of the PDE-models (Appendix E) is provided in two kinds of files: *scalar.data* and *soln****.data* (the asterisks stand for a four digit number). The file *scalar.data* is generated at the end of a simulation and contains the values of influx, total WIP, mean velocity and outflux for each time sample. A file *soln****.data* is generated at each time sample or at a selection of the time samples (to prevent an 'explosion' of output files), and contains the values of density, velocity and flux for all samples of x.

The development in time of the flow time can be determined from the influx and outflux data of *scalar.data*. By integrating the influx and outflux with respect to time, the amount of lots that has respectively entered and left the system is obtained. From these integrals, the time that the i^{th} lot has respectively entered and left the system can be determined. By taking the difference between these two times, the flow time is obtained for the i^{th} lot. This procedure is visualized in Figure F.1.

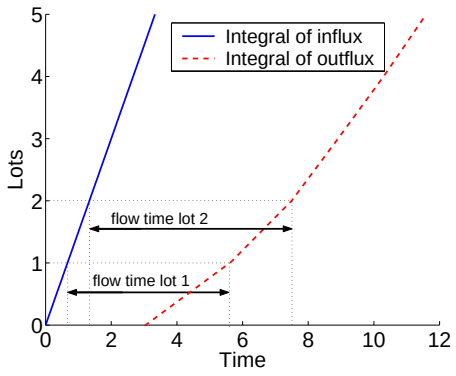


Figure F.1: Obtaining flow time from in- and outflux data

The development of the throughput in time can be found directly in *scalar.data*, since the outflux is in fact the throughput.

The development in time of the WIP per process can be derived from the density as a function of place and time in the files *soln****.data*. For each file the samples of x are divided into m equal parts, in which m denotes the number of processes in the system. Note that in order to do this, $nx - 1$ should be a multiple of m (nx is the number of discretization points in x-direction). Then, in each part the density is integrated with respect to x, using the trapezium rule, to obtain the WIP in that part.

When the development in time of the flow time, throughput, and WIP is obtained, all performance measures can be derived.

The described output processing is performed by the MATLAB® file *Process_cpp.m* for ramp up simulations and *Process_cppd.m* for ramp down simulations, which can be found on the CD-ROM that goes with this report. The MATLAB® file *Cppfilm.m* can also be found on this CD-ROM. This file can be used to plot the density, velocity and flux as a function of time and place in a movie, and is useful for examination of the PDE behaviour and the effect of fully discrete approximations. Furthermore, the CD-ROM contains the MATLAB® files *Showcpp_rampup.m* and *Showcpp_rampdown.m*, which can be used to show the useful data when the output processing already has been performed.

F.2 Output processing for the validation model

The DEM (Appendix D) generates output each time a lot leaves the system. At that moment, the current simulation time, flow time, inter departure time and WIP per process are written to the output file. Here, the inter departure time is defined as the time between the departure of two subsequent lots. This parameter is used to determine the throughput of the line, as is explained later in this appendix.

From the stochastic behaviour described by the DEM, the development in time of the mean flow time, mean inter departure time and mean WIP per process should be determined. In order to do so, several replications of a simulation are run [Law00]. The output samples across the replications are independent, so that mean (time dependent) values can be determined for all output parameters. For example, let φ_{11}, φ_{12}, ..., φ_{1j} be the flow time output of one simulation. Then, for a set of r replications the output is:

$$\begin{array}{cccc} \varphi_{11}, & \varphi_{12}, & \cdots, & \varphi_{1j} \\ \varphi_{21}, & \varphi_{22}, & \cdots, & \varphi_{2j} \\ \vdots & \vdots & & \vdots \\ \varphi_{r1}, & \varphi_{r2}, & \cdots, & \varphi_{rj} \end{array} \tag{F.1}$$

and the mean flow time of the i^{th} lot can be determined by taking the average of the i^{th} column. For the WIP per process and the inter departure time, the same procedure can be used.

In order to make sure that the determined mean values are correct, a confidence interval should be determined as well. However, since the output samples in a column of (F.1) are not random variables of a normal distribution, the confidence interval for a single set of replications is not trustworthy. If, on the other hand, more sets of replications are run, then by the *central limit theory* [Mon99] the mean values of the different sets approximately can be seen as random variables of a normal distribution, with which a reliable confidence interval can be determined.

The experiments with the DEM as described in Section 4.1 are thus performed as follows: sets of 100 simulations are run and of each set the mean values for the flow time, WIP per process and inter departure time are calculated. With these mean values of the sets, the mean values and the corresponding 95% two-sided confidence intervals of the 'normal distributions' are determined. New sets of simulations are performed until the confidence intervals of the mean flow time are smaller than 2% of the mean value. It is assumed that at that moment, the confidence intervals of the mean inter departure time and mean WIP per process are small enough as well.

The development in time of the mean flow time and the mean WIP per process are obtained directly from the method as described above. The development in time of the mean throughput is determined by taking the inverse of the calculated mean inter departure rate. With these data, all the performance measures can be derived.

The MATLAB® file *Runsimulation.m* that has been used to execute the method as described above, can be found on the CD-ROM that goes with this report. This CD-ROM also contains the MATLAB® files *Showresults_rampup.m* and *Showresults_rampdown.m* that can be used to calculate and plot the useful data when the method already has been executed.

Appendix G

Results of experiments: performance of PDE-models

In Chapter 4, the performance of the PDE-models is investigated by means of experiments. The results of these experiments are given in this appendix.

RAMP UP: m=10; μ=2.0; nx=101; *cflcond*=0.5

u = 25% ($t = 50$)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	6.7	23 (8 lots)	5.0	0.5	35	0.33	31
Model 1	70 sec	6.7	15 (4 lots)	6.2	0.5	17	0.33	17
Model 2	70 sec	6.7	23 (8 lots)	5.5	0.5	24	0.33	25
Model 3	80 sec	6.7	11 (2 lots)	6.6	0.5	10	0.33	10

u = 50% ($t = 100$)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	10	52 (42 lots)	5.0	1.0	50	1.0	54
Model 1	100 sec	10	35 (25 lots)	7.1	1.0	36	1.0	38
Model 2	110 sec	10	57 (47 lots)	5.5	1.0	51	1.0	64
Model 3	90 sec	10	19 (9 lots)	7.6	1.0	22	1.0	22

$u = 75\%$ ($t = 300$)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	20	211 (287 lots)	5.0	1.5	130	3	221
Model 1	80 sec	20	143 (184 lots)	8.2	1.5	121	3	151
Model 2	90 sec	20	231 (317 lots)	5.5	1.5	168	3	254
Model 3	205 sec	20	74 (81 lots)	7.8	1.5	82	3	87

Remark:

- For Model 3, *cflcond* = 0.5 is not small enough (see general remark at the end of this page). Therefore *cflcond* = 0.2 is used.

$u = 95\%$ ($t = 5000$)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	100	4863 (9050 lots)	5.0	1.9	1400	19	3839
Model 1	170 sec	101	3421 (6311 lots)	9.1	1.9	1513	19	3477
Model 2	300 sec	100	5747 (10733 lots)	5.5	1.9	1883	19	5872
Model 3	764 sec	101	1790 (3212 lots)	7.9	1.9	1442	19	2166

Remarks:

- For Model 2, $t = 5000$ is too short too reach steady state, therefore $t = 7000$ is used.
- For Model 3, even *cflcond* = 0.2 is not small enough (see general remark at the end of this page). Therefore, *cflcond* = 0.1 is used.

GENERAL REMARKS FOR RAMP UP SIMULATION WITH 10 MACHINES IN LINE

- As an indication, it is noted that the total simulation time for the validation model is in the order of hours (depending on the simulation length and required confidence interval).
- Model 2 gives a 'step-by-step' development of the throughput.
- For Model 3, there is a border region for *cflcond* above which a strange curve appears in the throughput plot and results are very dependent on *cflcond*. Below this border region, results are more accurate and remain approximately independent of *cflcond*. In the border region itself, there occurs a singularity (negative flow time).
- Especially for low utilizations ($< 25\%$), φ_{ss} can be made more accurate by increasing nx.
- Since the development of δ_{ss} for the validation model is not very smooth, the value for *Time to* δ_{ss} calculated by the MATLAB® file *showresults_rampup.m* is not always accurate; in these cases a visual estimation is made.

RAMP UP: m=50; μ=2.0; nx=101; *cflcond*=0.5

u = 25% (t = 200)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	33.3	69 (18 lots)	25	0.5	61	0.33	75
Model 1	50 sec	33.6	71 (19 lots)	29	0.5	83	0.33	84
Model 2	50 sec	33.6	107 (37 lots)	24	0.5	118	0.33	127
Model 3	35 sec	33.6	45 (6 lots)	30	0.5	51	0.33	51

u = 50% (t = 300)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	50	119 (70 lots)	25	1.0	140	1.0	140
Model 1	40 sec	50	170 (120 lots)	32	1.0	179	1.0	193
Model 2	50 sec	50	287 (237 lots)	21	1.0	259	1.0	324
Model 3	20 sec	50	95 (45 lots)	31	1.0	110	1.0	114

Remark:

- For Model 2, t = 300 is too short to reach steady state, therefore t = 500 is used.

u = 75% (t = 700)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	100	552 (677 lots)	25	1.5	550	3	575
Model 1	40 sec	100	581 (724 lots)	37	1.5	525	3	655
Model 2	60 sec	99	1015 (1376 lots)	19	1.5	792	3	1157
Model 3	70 sec	101	367 (401 lots)	31	1.5	410	3	452

Remark:

- For Model 2, t = 700 is too small to reach steady state, therefore t = 1200 is used.
- For Model 3, *cflcond* = 0.5 is not small enough (see general remark at the end of this page). Therefore, *cflcond* = 0.2 is used.

$u = 95\%$ ($t = 12000$)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	500	11423 (20775 lots)	25	1.9	5000	19	11002
Model 1	184 sec	500	14732 (27051 lots)	40	1.9	7259	19	15082
Model 2	325 sec	>479	>18444 (34144 lots)	18	1.9	8148	>18	>19228
Model 3	380 sec	505	8932 (16021 lots)	31	1.9	7172	19	11245

Remarks:

- For Model 1, $t = 12000$ is too short to reach steady state, therefore $t = 18000$ is used.
- For Model 2, $t = 12000$ is too short too reach steady state, therefore $t = 20000$ is used, which is still too short...
- For Model 3, even *cflcond* $= 0.2$ is not small enough (see general remark at the end of this page). Therefore, *cflcond* $= 0.1$ is used.

GENERAL REMARKS FOR RAMP UP SIMULATION WITH 50 MACHINES IN LINE

- As an indication, it is noted that the total simulation time for the validation model is in the order of hours (depending on the simulation length and required confidence interval).
- Model 2 gives a 'step-by-step' development of the throughput.
- For Model 3, there is a border region for *cflcond* above which a strange curve appears in the throughput plot and results are very dependent on *cflcond*. Below this border region, results are more accurate and remain approximately independent of *cflcond*. In the border region itself, there occurs a singularity (negative flow time).
- Especially for low utilizations ($< 25\%$), φ_{ss} can be made more accurate by increasing nx.
- By increasing nx, the flow time of the first lot becomes more accurate for Model 2 (for Models 1 and 3 this does not have a large influence, in fact: the results get worse).
- Since the development of δ_{ss} for the validation model is not very smooth, the value for *Time to* δ_{ss} calculated by the MATLAB® file *showresults_rampup.m* is not always accurate; in these cases a visual estimation is made.

RAMP DOWN: m=10; μ=2.0; nx=501; $cflcond$=0.5

u = 50% to: u = 25% (t = 50) ($\rho_{init} = 10$)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	6.7	21 (17 lots)	7.7	0.5	21	0.33	27
Model 1	100 sec	6.7	21 (7 lots)	8.1	0.5	22	0.33	21
Model 2	200 sec	6.7	31 (12 lots)	9.5	0.5	31	0.33	35
Model 3	170 sec	6.6	15 (4 lots)	9.0	0.5	13	0.33	13

u = 75% to: u = 25% (t = 100) ($\rho_{init} = 30$)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	6.7	49 (51 lots)	17	0.5	45	0.33	47
Model 1	280 sec	6.6	31 (12 lots)	13	0.5	32	0.33	32
Model 2	130 sec	6.6	51 (22 lots)	19	0.5	52	0.33	55
Model 3	210 sec	6.6	29 (11 lots)	19	0.5	27	0.33	27

u = 90% to: u = 25% (t = 200) ($\rho_{init} = 90$)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	6.7	101 (137 lots)	46	0.5	103	0.33	105
Model 1	250 sec	6.3	54 (24 lots)	31	0.5	56	0.33	56
Model 2	350 sec	6.3	98 (46 lots)	49	0.5	98	0.33	99
Model 3	450 sec	6.3	68 (31 lots)	49	0.5	67	0.33	67

$u = 95\%$ to: $u = 25\%$ ($t = 300$) ($\rho_{init} = 190$)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	6.7	185 (279 lots)	95	0.5	176	0.33	197
Model 1	360 sec	5.8	90 (42 lots)	60	0.5	91	0.33	91
Model 2	220 sec	5.9	168 (81 lots)	99	0.5	170	0.33	171
Model 3	490 sec	5.9	134 (64 lots)	99	0.5	134	0.33	134

GENERAL REMARKS FOR RAMP DOWN SIMULATION WITH 10 MACHINES IN LINE

- As an indication, it is noted that the total simulation time for the validation model is in the order of hours (depending on the simulation length and required confidence interval).
- For Model 1, the throughput first increases before it decreases to the steady state value.
- Model 2 gives a 'step-by-step' development of the throughput.
- Model 3 has a very steep convergence to the new steady state (flow time and throughput).
- Especially for low utilizations ($< 25\%$), φ_{ss} can be made more accurate by increasing nx.

RAMP DOWN: m=50; μ=2.0; nx=501; $cflcond$=0.5

u = 50% to: u = 25% (t = 200) (ρ_{init} = 50)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	33.3	82 (74 lots)	46	0.5	85	0.33	94
Model 1	150 sec	33.2	98 (32 lots)	43	0.5	108	0.33	109
Model 2	90 sec	33.3	150 (58 lots)	50	0.5	157	0.33	175
Model 3	120 sec	33.2	67 (17 lots)	49	0.5	67	0.33	67

u = 75% to: u = 25% (t = 300) (ρ_{init} = 150)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	33.3	170 (218 lots)	95	0.5	168	0.33	187
Model 1	60 sec	32.8	151 (59 lots)	72	0.5	162	0.33	164
Model 2	50 sec	32.9	251 (109 lots)	99	0.5	257	0.33	275
Model 3	150 sec	32.8	135 (51 lots)	99	0.5	134	0.33	134

u = 90% to: u = 25% (t = 600) (ρ_{init} = 450)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	33.3	382 (625 lots)	243	0.5	380	0.33	412
Model 1	130 sec	31.4	268 (118 lots)	161	0.5	278	0.33	281
Model 2	200 sec	31.7	484 (226 lots)	249	0.5	490	0.33	499
Model 3	300 sec	31.6	334 (151 lots)	249	0.5	335	0.33	335

$u = 95\%$ to: $u = 25\%$ ($t = 1000$) ($\rho_{init} = 950$)	Computer time	φ_{ss}	Time to φ_{ss}	φ 1st lot	δ_{ss}	Time to δ_{ss}	w_{ss} (per process)	Time to w_{ss}
Validation model	- -	33.3	759 (1312 lots)	491	0.5	734	0.33	783
Model 1	180 sec	28.9	447 (209 lots)	309	0.5	256	0.33	458
Model 2	140 sec	29.7	840 (405 lots)	500	0.5	851	0.33	858
Model 3	200 sec	29.6	670 (320 lots)	499	0.5	669	0.33	671

GENERAL REMARKS FOR RAMP DOWN SIMULATION WITH 50 MACHINES IN LINE

- As an indication, it is noted that the total simulation time for the validation model is in the order of hours (depending on the simulation length and required confidence interval).
- For Model 1, the throughput first increases before it decreases to the steady state value.
- Model 2 gives a 'step-by-step' development of the throughput.
- Model 3 has a very steep convergence to the new steady state (flow time and throughput).
- Especially for low utilizations ($< 25\%$), φ_{ss} can be made more accurate by increasing nx.

Appendix H

Nonlinear MPC

In this appendix, the concept of nl-MPC is explored further. In Appendix H.1, the concept of nl-MPC is described and it is explained how the design parameters of nl-MPC can be tuned in order to obtain a properly working controller [Ess02b]. In Appendix H.2 a part of the code of the nl-MPC implementation in MATLAB® is given. This code is explained in Appendix H.3.

H.1 The concept of (nonlinear) MPC

In this appendix, first the general concept of (nonlinear) MPC and the underlying concept of the moving horizon are treated. After that, the tuning of the design parameters of nl-MPC is discussed. For further information on (nonlinear) MPC, the interested reader is referred to the lecture notes on MPC [Ess02b], of which a copy is included on the CD-ROM that goes with this report.

Figure H.1 gives a schematic visualization of the concept of (nonlinear) MPC. For simplicity, a single-input-single-output system is considered. At the current sample k, the output $y(k|k)$ is known and the future outputs $(y(k+1|k), y(k+2|k), \ldots, y(k+p|k))$ are predicted for the length of the prediction horizon p using an internal approximation model of the considered process. Here, $y(k+i|k)$ represents the predicted value of the output y at sample $k+i$ based on the information available at sample k.

The future values of the input $(u(k|k), u(k+1|k), \ldots, u(k+m-1|k))$ over the control horizon with a length of m samples are determined by minimizing the value of the goal function J. Often, this goal function is a weighted summation of the deviations between the predicted outputs y and the desired reference trajectory y_{ref}:

$$J(\mathbf{u}) = \sum_{i=1}^{p} \|(y(k+i) - y_{\mathrm{ref}}(k+i))\|_{\mathbf{Q}}^2 \,, \tag{H.1}$$

in which $\mathbf{u}$ is the vector of future inputs over the control horizon and $\|x\|^2_{\mathbf{Q}} = x^T\mathbf{Q}x$, in which $\mathbf{Q}$ is a weighting matrix. Note that for nonlinear MPC, the output vector $\mathbf{y}$ is a nonlinear function of the input vector $\mathbf{u}$ and thus the minimization of the goal function is a nonlinear optimization problem.

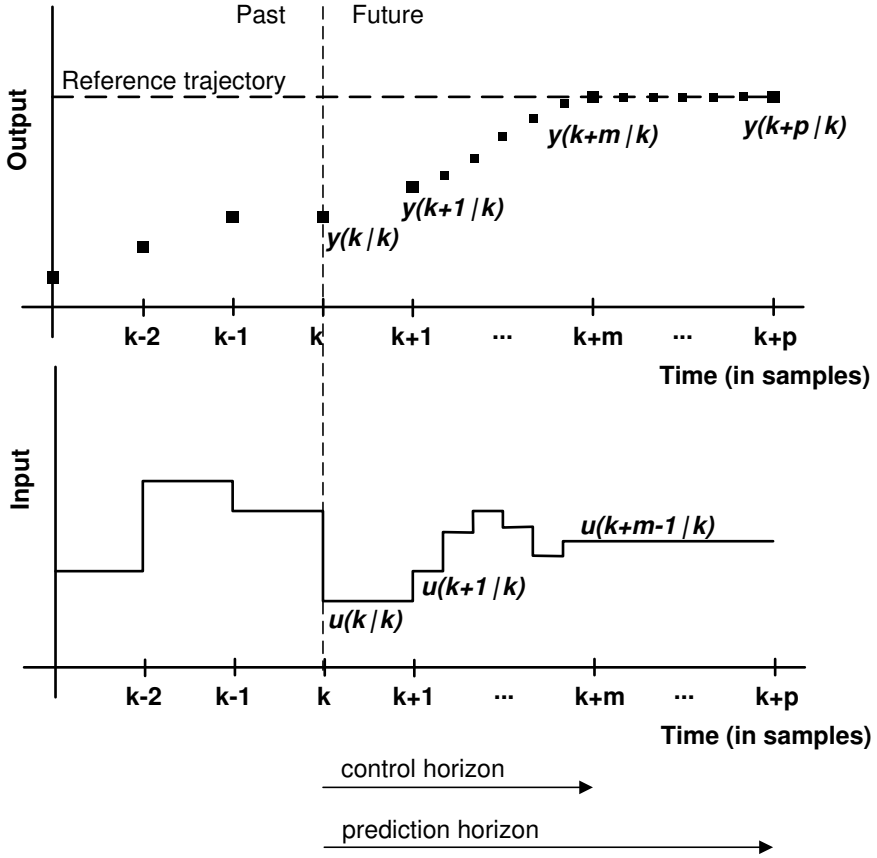


Figure H.1: Concept of Model Predictive Control

Now, the concept of the moving horizon is considered. Once an optimal set of future inputs has been determined, the first of these inputs, $u(k|k)$, is applied to the real process. At the next sample the output value of the process is measured, after which the prediction and optimization are performed for this new sample. The horizon has 'moved' with one sample. By this concept of the moving horizon, (nonlinear) MPC can be used as a real time controller for dynamic processes. A restriction to the use of (nonlinear) MPC, however, is that the computer time that is required to find an optimal set of future inputs should be negligible in comparison to the sample time. In case the computer time is not negligible, but is still smaller than one sample time, an adapted version of (nonlinear) MPC can still be used [Ess02b]. This version, however, is not considered further in this report.

Parameter tuning

In order to obtain a properly working controller, the parameters in (nonlinear) MPC should be well-tuned. The most important parameters are presented here and the way to tune them is discussed. The considered parameters are:

- The sample time dt
- The length (or number of samples) of the prediction horizon p
- The length (or number of samples) of the control horizon m
- The blocking factor *blfactor*
- The weighting matrix $\mathbf{Q}$

The sample time should be chosen with respect to the process dynamics, the input, and the reference trajectory. The sample time should be chosen small enough so that the process dynamics and reference trajectory can be accurately described and that the input signal can be changed fast enough to obtain a desired effect in the process. On the other hand, a smaller sample time causes a larger computer time. It can therefore be concluded that MPC can not be used for relatively 'fast' processes, since usage of the required sample time would result in a computer time that is not negligible in comparison to the sample time.

For the length of the prediction and control horizon, the following guidelines are given. The prediction horizon should be long enough to monitor the entire effect of the (future) inputs that are determined at a sample. This means that the prediction horizon must exceed the largest controlled time constant of the process, periods of inverse response, and dead time periods. Often, the control horizon is chosen to be 1/6 to 1/3 of the prediction horizon, depending on the kind of process. Note that a longer control horizon is more 'aggressive' and therefore more suitable for reference trajectories that vary relatively fast in time. The upper limit of the control- and prediction horizon lengths is again determined by the computer time, since larger horizons require more computer time.

In cases that a large control horizon is required, but leads to unacceptable computer times, blocking can be a solution. For control blocking, the control horizon is divided into a number of blocks, that consist of *blfactor* samples. Within a control block, the input is not allowed to change. In this way, the length of the control horizon is preserved, while the computer time is reduced (the number of design variables $\mathbf{u}$ is reduced by a factor *blfactor*). A disadvantage of control blocking, however, is that it makes the controller less 'aggressive'. A similar blocking technique can also be applied to the prediction horizon.

Finally, the weighting matrix can be used to emphasize certain aims in the goal function. Consider for example the goal function in (H.1). If the weighting matrix here

equals the identity matrix, the deviation between output and reference weights equally for all samples in the prediction horizon. If, however, the values on the diagonal of the weighting matrix increase for increasing row numbers, then the deviation between output and reference weights heavier for later samples in the prediction horizon. In this way, it is emphasized that especially *towards the end* of the prediction horizon, the output should be equal to the reference value. The output values in the early samples of the prediction horizon are not so important then.

H.2 Nl-MPC code

The code of the implementation of nl-MPC in MATLAB® is subdivided over eight files. In Appendix H.3, an overview is given of the functions of each of these files. Here, only the code of the three most important files is given, i.e., *nlmpc.m*, *Process.m* and *FUN.m*. An explanation on these files can be found in Appendix H.3 as well. On the CD-ROM that goes with this report all files that are related to the implementation of nl-MPC are included.

nlmpc.m

This is the main file of the implementation of nl-MPC. From this file, all other files are called.

```
clear all
close all

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%  Nonlinear Model Predictive Control for Discrete Time Systems  %
%%                      Version 09-12-2003                        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Requirements:
% Matlab 5.3 (or higher) with Optimization Toolbox (fmincon.m)
%
% Features:
% - Model predictive control for non-linear discrete time systems
% - Weighting factors for input change and output correctness (optional)
% - Constraints on input, input change and output (optional)
% - Exponential increasing weighting factor for output correctness (optional)
% - Final state weighting (optional)
% - Full sample computational delay (optional)
% - Error checking of input values
% - Block samples for prediction and control horizon (3-12-2003)
% - Non model based observer: first order output disturbance
%   observer (8-12-2003)
% - Intermediate plots of prediction horizon and comparison with earlier
%   samples (9-12-2003)
% - Constraint on outflux (optional): no positive backlog may occur (23-12-2003)
%
% Created by Roel van den Berg, B.Sc.
% Eindhoven University of Technology
```

```
% Department of Mechanical Engineering, Systems Engineering Group
%
% This file is based on the linear mpc file (version 8-4-02),
% created by Harm van Essen, Dynamics and Control Technology
% Group, TUE.

%%%%%%%%%%
% INPUTS %
%%%%%%%%%%

% General program inputs
pcdelay = 0;  % full (block) sample computational delay (0 = OFF, 1 = ON)
horizon = 0;  % view of predictionhorizon during simulation every
              % xth sample (0 = OFF, 1 = every block sample, 2 = once per
              % 2 block samples, etc.)

% Definition of related m-files
process     = 'Process';  % functionfile with nonlinear process model
prediction  = 'Process';  % functionfile with predictionmodel
setpoint    = 'Setpoint'; % functionfile with setpoint

% Definition of fmincon options (see OPTIMSET)
% "optimset('fmincon')" displays all default values for fmincon
OLDOPTS = [];
OLDOPTS = optimset(OLDOPTS,'Largescale','off');   % {on}/off, use of largescale algorithm
OLDOPTS = optimset(OLDOPTS,'Display','off');      % off/iter/{final}
%OLDOPTS = optimset(OLDOPTS,'MaxIter',4000);      % positive integer
%OLDOPTS = optimset(OLDOPTS,'MaxFunEvals',10000); % positive integer
Options = optimset(OLDOPTS);

% Number of buffers in line
nn = 10;

% Description of states and input for graphical output
utxt = {['input: arrival rate']};
xtxt = []; xref = []; xcon = []; yptxt = [];
for i = 1 : nn
xtxt = [xtxt; {['WIP machine ' num2str(i)]}];
xref = [xref; {['reference WIP ' num2str(i)]}];
xcon = [xcon; {['constraint WIP ' num2str(i)]}];
yptxt = [yptxt; {['Real WIP machine' num2str(i)]}];
end

% Initial state and initial input values
x_init = 0*ones(nn,1);       % initial WIP machines
u_init = [0];                % initial input (arrival rate)

% Start and end time of simulation (in samples)
ts = 0;  te = 300; % start- and endtime (in hours)
ks = ts/0.25;      % first sample
ke = te/0.25;      % last sample (0.25 is length of one sample, see Process.m)

% Prediction and controlhorizon (in samples)
pt = 150; mt = 8;  % prediction- and controlhorizon (in hours)
blfactor = 4;      % Blocking factor for control- and prediction horizon (positive integer)
                   % for blfactor > 1, control and prediction horizon are
                   % calculated only once per 'blfactor' samples. During a
                   % block, the input remains constant.

p = pt/0.25/blfactor;  % prediction horizon (in block samples)
m = mt/0.25/blfactor;  % control horizon (in block samples)
```

```
% Weighting factors
R_du = [1];                  % weighting factor for input changes du in MPC cost function
Q_y  = [zeros(nn-1,1); 1];   % [0; 0; 1];
                             % weighting factors for controlled outputs in MPC cost
                             % function (0 = NOT controlled)
                             % NOTE: by default, yk = Ck * xk, where Ck has only '0' and
                             % '1' elements. For a more complex yk, this should be
                             % changed in the first part of the initialization.

% Constraints
u_du  = [2];       % maximal step size per sample of each input
u_min = [0];       % minimal absolute value u of each input (-Inf is minus infinity)
u_max = [2];       % maximal absolute value u of each input (Inf is inifinity)

Yconstraints = zeros(size(x_init));    % selected constrained outputs (0 = OFF, 1 = ON)
Y_min = 0*ones(size(x_init));          % minimal absolute value Y of each output
Y_max = 1000*ones(size(x_init));       % maximal absolute value Y of each output
                                       % WARNING: do not use Inf or -Inf for the
                                       %          constraints, use a large number instead
                                       % NOTE: by default, ycon = Ccon * xk, where Ccon
                                       %       has only '0' and '1' elements. For a more
                                       %       complex ycon, this should be changed in
                                       %       the first part of the initialization.
                                       % NOTE: Because of a nonnegative input and
                                       %       nonnegative initial state, all states in
                                       %       future samples will be automatically
                                       %       nonnegative. Therefore, a constraint that
                                       %       guarantees nonnegative output is superfluous.

% "No positive backlog"-constraint on outflux  (negative backlog represents a safety stock)
q_constraint = 0;  % outflux constraint (0 = OFF, 1 = ON)
b_init = 0.0;      % NOTE: only applicable for steady state situations that change to
                   %       another steady state AFTER A WHILE (so tswitch /= 0)
                   %       For ramp-up, an extra amount of products should be made before
                   %       tswitch to prevent a shortage of products
                   %       For ramp-down, the outflux may not decrease until after tswitch


% Extra features for terminal state improvement
cresfactor = 1.05;           % exponential increasing setpoint weight over prediction
                             % horizon (1 = no additional penalty)
                             % NOTE: In choosing a value for cresfactor one
                             %       should make sure that the value of the
                             %       goalfunction does not 'explode'
FWy = zeros(size(x_init));   % final state weighting (terminal state penalty)
                             % (1 = no additional penalty)


% Gain for non-model based first order observer (filtering noise from measured data)
 gain_d = 0.5;     % filtergain output disturbance filter (scalar value)
                   % d(k+1|k) = d(k|k-1) + Kd ( yp(k) - ym(k|k-1) )
                   % d has the dimension of full state, but remains zero
                   % for unmeasured outputs (Kd is of size (nx by nx))
                   % 0 <= Kd <= 1, for Kd closer to 1: more aggressive
                   % yp(k) = real value (measured from real process)
                   % ym(k|k-1) = meainternal model output

%%%%%%%%%%%%%%%%%
% END OF INPUTS %
%%%%%%%%%%%%%%%%%
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% CHECK INPUT FOR ERRORS AND DISPLAY SETTINGS %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

xk = x_init; nx = max(size(xk));  % number of states
uk = u_init; nu = max(size(uk));  % number of inputs

% Check ks, ke, m, p and blfactor
if ks - floor(ks) ~= 0 | ks < 0 | ke - floor(ke) ~= 0 | ke < ks
    error('ks and ke [in samples] should be nonnegative integers and ks < ke');
end
if m - floor(m) ~= 0 | m < 0 | p - floor(p) ~= 0 | m>p
    error('m and p [in samples] should be nonnegative integers, m <= p');
end
if blfactor - floor(blfactor) ~= 0 | blfactor < 1
    error('blocking factor should be a positive integer');
end


% Check the initial input and initial state if they are within the constraints
if u_init < u_min | u_init > u_max
    error('The initial input value is not within the defined constraints')
elseif min(Yconstraints.*(x_init - Y_min)) < 0 | max(Yconstraints.*(x_init - Y_max)) > 0
    error('The initial state is not within the defined constraints')
end

% Check if the initial input and initial state are nonnegative
if u_init < zeros(size(u_init))
    error('The initial input value should be nonnegative')
elseif min(x_init-zeros(size(x_init))) < 0
    error('The initial state should be nonnegative')
end

% Check if variable pcdelay is either 0 or 1
if pcdelay ~= 0 & pcdelay ~= 1 error('Variable pcdelay should be either 0 or 1'); end

% Check if variable horizon is a nonnegative integer
if horizon - floor(horizon) ~= 0 | horizon < 0
    error('Variable horizon should be a nonnegative integer');
end

% Check if initial backlog is not positive if q_constraint = 1
if q_constraint ~= 0 & b_init > 0.0
    error('Backlog should be nonpositive if constraint on backlog is on');
end

% Check if gain_d is scalar and value is between 0 and 1
if max(size(gain_d)) ~= 1 error('Variable gain_d should be a scalar'); end
if gain_d > 1 | gain_d < 0 error('Variable gain_d should be in the interval [0,1]'); end
if gain_d == 0 disp('WARNING: observer disabled'); end

% Check size of matrices
if max(size(Q_y))~=nx
    error('Q_y does not contain the same number of elements as states');
end
if max(size(FWy))~=nx
    error('FWy does not contain the same number of elements as states');
end
if (max(size(R_du))+max(size(u_du))+max(size(u_min))+max(size(u_max))~=4*nu)
    error('Matrices for inputs do not contain the same number of elements as inputs');
end
```

```
if (max(size(Yconstraints))+max(size(Y_min))+max(size(Y_max))~=3*nx)
    error('Matrices outputs do not contain the same number of elements as states');
end

disp(' ')
disp('Nonlinear Model Predictive Control for a manufacturing system')
disp(' ')
disp(['first sample is: k = ' num2str(ks)])
disp(['last sample is : k = ' num2str(ke)])
disp(' ')
disp(['prediction horizon is : ' num2str(p) ' [block samples]'])
disp(['control horizon is    : ' num2str(m) ' [block samples]'])
disp(['   (each block consists of ' num2str(blfactor) ' samples)'])
disp('   ');
disp(['   the process model is                          : ', process, '.m'])
disp(['   the predictionmodel is                        : ', prediction, '.m' ])
disp(['   the setpoint generation is in the file        : ', setpoint,'.m' ])
disp('   ');
disp(['   observer gain is : ' num2str(gain_d)]);
disp('   ');
if q_constraint==1
    disp(['   constraint on backlog is ON: no positive backlog may occur']);
    disp('   ');
end
if pcdelay==1
    disp(['full sample computational delay is : ON']);
else
    disp(['full sample computational delay is : OFF']);
end
if horizon~=0
    disp(['view of predictionhorizon is   : ON, every ' num2str(horizon) ' [samples].']);
else
    disp(['view of predictionhorizon is   : OFF']);
end
disp('  ');


%%%%%%%%%%%%%%%%%%
% INITIALIZATION %
%%%%%%%%%%%%%%%%%%

% Initialization of Ccon matrix: ycon = Ccon * xk
if max(Yconstraints) == 0 & q_constraint == 0    % no output constraints
    y_min = []; y_max = [];
    Ccon = []; nonlcon = [];
elseif max(Yconstraints) == 0 & q_constraint ~= 0
    y_min = []; y_max = [];
    Ccon = []; nonlcon = 'NONLCON';
else max(Yconstraints) ~= 0
    y_min = []; y_max = []; Ccon = []; Cplot = [];
    for i = 1 : nx
        if Yconstraints(i) ~= 0
            cc = zeros(1,nx); cc(1,i) = 1; Ccon = [Ccon;cc]; Cplot = [Cplot, i];
            y_min = [y_min; Y_min(i)];
            y_max = [y_max; Y_max(i)];
        end
    end
    ycon = Ccon*xk;          % column of CONSTRAINED outputs
    ncon = max(size(ycon));  % number of CONSTRAINED outputs
    nonlcon = 'NONLCON';
end
```

```
% Initialization of Ck matrix: yk = Ck * xk
Ck = []; Xplot = [];
for i = 1 : nx
    if Q_y(i) ~= 0
        cc = zeros(1,nx); cc(1,i) = 1; Ck = [Ck; cc]; Xplot = [Xplot, i];
    end
end
yk = Ck*xk;                 % column of CONTROLLED outputs
ny = max(size(yk));         % number of CONTROLLED outputs


% Initialization of Kd matrix, d_init and yp
Kd = gain_d*eye(length(yk)); % output disturbance filter gain matrix (measured
                            % states are equal to the controlled states)
d_init = zeros(size(yk));   % initial output disturbance
yp = Ck*x_init;             % real output vector at sample ks = 0

% Initialization of setpoint or reference trajectory at first sample ks
[ykref,bkref] = Setpoint(ks,1,x_init,Ck);

% Store initial data for visalization of results
ttot = ks;        % Sample ks = 0
xtot = xk';       % State at sample ks = 0
if pcdelay == 0
    utot = [];    % In case of no computational delay, the first input (applied at ks = 0)
                  % is also calculated by fmincon
else
    utot = uk';   % In case of a full sample computational delay, the first calculated
                  % input can only be applied at ks+1
end               % so the defined initial input is applied at ks = 0
ytot = yk';       % Output at sample ks = 0
yreftot = ykref'; % Reference values for output at sample ks = 0
breftot = bkref'; % Reference values for backlog at sample ks = 0
dtot = d_init';   % Output disturbance vector at sample ks = 0
btot = b_init;    % Backlog at sample ks = 0
yptot = yp';      % Real output at sample ks = 0

% Initialization of cputime, so that computation time can be determined
cput = cputime;

% Initialization of 'previous' values
xkold = xk;        % State at sample ks = 0
ukold = uk;        % Input at sample ks = 0 (NOT USED FOR pcdelay == 0)
dold  = d_init;    % Output disturbance at sample ks = 0
bold  = b_init;    % Backlog at sample ks = 0
ypold = yp;        % Real output at sample ks = 0

%%%%%%%%%%%%%%%%%%%%%%%%%
% END OF INITIALIZATION %
%%%%%%%%%%%%%%%%%%%%%%%%%



%%%%%%%%%%%%%%
% SIMULATION %
%%%%%%%%%%%%%%

% Define constant matrices for use in mpc goalfunction and constraints
[FW,QQ,RRa,Lb,Ub,AA,B,Y_maxmin] = Matrices_constant(nu,nx,p,m,Q_y,FWy,cresfactor,R_du,
                                                    u_min,u_max,u_du,y_min,y_max);
AA = AA(:,nu+1:end);   % Adaption of AA (see for more info 'Matrices_constant.m')
```

```
% Definition of start values for the nonlinear search for optimal input values
U0 = 0*ones(m*nu,1); % Note: for time varying reference, U0 can be different for
                     % pcdelay == 0 and pcdelay ~= 0!

FVAL = [];

% Start simulation control loop: steps of 1 from ks -> ke
for ki = ks+blfactor : blfactor :  ke + (blfactor-1)
    disp(['sample ' num2str(ki) ' (block sample ' num2str(ki/blfactor) ') ,
           time = ' num2str(ki*0.25) ]);

    % Adaption of B (see for more info 'Matrices_constant.m')
    B(1:nu,1) = u_du + ukold;
    B(m*nu+1:(m+1)*nu,1) = u_du - ukold;


    % Determine reference values for output as inspected by fmincon
    if pcdelay == 0
        [yref,bbref] = Setpoint(ki,p,x_init,Ck);
        yrefold = [];
    else
        [yref,bbref] = Setpoint(ki+1,p,x_init,Ck);
        yrefold = Setpoint(ki,1,x_init,Ck);
    end

    % Here the output yp(k) of the real process of sample k should be known,
    % so that the prediction of d(k+1|k) (here: d) can be made.
    d = dold + Kd*(ypold - Ck*xkold - dold);

    % Find fmincon optimal input values
    [U,fval,exitflag,output] = fmincon('FUN',U0,AA,B,[],[],Lb,Ub,nonlcon,Options,FW,QQ,RRa,
                                       Y_maxmin,yref,yrefold,ny,Ck,Ccon,pcdelay,p,m,
                                       blfactor,q_constraint,xkold,ukold,d,bold,bbref);
    FVAL = [FVAL;fval];

    % Interprete output of fmincon
    if exitflag < 0
        disp(['Infeasible solution at sample ' num2str(ki)]);
        % Actually, exitflag < 0 means that fmincon did not converge to a
        % solution. Most of the time, however, this means that the given
        % output is not within the constaints, and therefore the solution
        % is infeasible.
    elseif exitflag == 0
        disp(['Maximum number of function evaluations was reached']);
    end

    % Determine values of xk, yk, uk, ykref at this sample ki
    if pcdelay == 0
        ukold = U(1:nu,1);           % This is the input at the PREVIOUS sample ki-1
        xk = Process(xkold,ukold,blfactor);
        yk = Ck*xk+d;
        [ykref,bkref] = Setpoint(ki,1,x_init,Ck);
        b = bold + Process(xkold,ukold,blfactor,ykref);
        yp = Ck*feval(prediction,xkold,ukold,blfactor);
    else
        uk = U(1:nu,1);              % This is the input at the CURRENT sample ki
        xk = Process(xkold,ukold,blfactor);
        yk = Ck*xk+d;
        [ykref,bkref] = Setpoint(ki,1,x_init,Ck);
        b = bold + Process(xkold,ukold,blfactor,ykref);
        yp = Ck*feval(prediction,xkold,ukold,blfactor);
    end
```

```
    % Store values for visualization of results
    ttot = [ttot; ki];           % Sample ki is added
    xtot = [xtot; xk'];          % State at sample ki is added
    if pcdelay == 0              % Input of the PREVIOUS sample ki-1 is added (which has
        utot = [utot; ukold'];   % influenced the state at the CURRENT sample ki)
    else                         % Input at CURRENT sample ki is added (which will
        utot = [utot; uk'];      % influence only the state at the NEXT sample ki+1)
    end
    ytot = [ytot; yk'];          % Output at sample ki is added
    yreftot = [yreftot; ykref']; % Reference values for output at sample ki are added
    breftot = [breftot; bkref']; % Reference values for backlog at sample ki are added
    dtot = [dtot; d'];           % Output disturbance at sample ki is added
    btot = [btot; b];            % Backlog at sample ki is added
    yptot = [yptot; yp'];        % Real output at sample ki is added


    % Update values xkold, ukold and start guess U0
    xkold = xk;
    dold = d;
    bold = b;
    ypold = yp;
    if pcdelay == 1
        ukold = uk;  % Since for pcdelay = 0, uk is not used, and ukold is already defined
    end
    U0 = [U(nu+1:end,1);U(end-nu+1:end,1)];


    % Display intermediate results of outputs, constraints and inputs
    % over the prediction horizon
    if horizon~=0 & mod(ki,blfactor*horizon) == 0
        plothorizon
        disp(['prediction horizon at sample : ' num2str(ki) ', press enter to continue'])
        newhori = input('or enter the number of block samples
                         untill the next print-horizon (0 = stop)');
        if isempty(newhori)
            disp(['Next horizon display in  ' num2str(horizon) '  block sample(s).']);
        elseif newhori==0
            horizon=0;
        elseif newhori>0
            horizon=newhori;
            disp(['Next horizon display in ' num2str(horizon) '  block sample(s).']);
        end
    end


end

%%%%%%%%%%%%%%%%%%
% END SIMULATION %
%%%%%%%%%%%%%%%%%%

% Display computation time and results
disp(['Computation time is ' num2str(cputime-cput) ' [s] for ' num2str(ke-ks)
       ' [samples] or ' num2str(0.25*(ke-ks)) ' [hours] real time'])
plotresults
save testresults
```

Process.m

The process description that was used for the first and second control problem as defined in Section 5.2 can be found in the following code.

```
function [OUT] = Process(WIP,uk,blfactor,varargin)

% nonlinear discrete time representation of the manufacturing process
% obtained by discretization of the PDE in time and place

% parameters
m = max(size(WIP)); % number of machines
xk = m*WIP;
dx = 1/m;
muu = 2; cflcond = 0.5;
dt = cflcond*dx*m/muu; %dt = 0.5/2 = 0.25 sec = 1 sample
cfl = dt/dx;

if nargin == 4
    demand_outflux = muu*(varargin{1})/(1+varargin{1});
    backlog = 0.0;
    for bb = 1 : blfactor
        % non-linear equations:
        xk_next(1,1) = xk(1) + cfl*(-muu*(xk(1)/m)/(1+(xk(1)/m)) + uk);

        for i = 2:length(xk)
            xk_next(i,1) = xk(i) + cfl*muu*(-(xk(i)/m)/(1+(xk(i)/m)) + (xk(i-1)/m)/(1+(xk(i-1)/m)));
        end
        actual_outflux = muu*(xk(m)/m)/(1+(xk(m)/m));
        backlog = backlog + dt*(demand_outflux - actual_outflux);
        xk = xk_next;
    end
    OUT = backlog;
else

    for bb = 1 : blfactor
        % non-linear equations:
        xk_next(1,1) = xk(1) + cfl*(-muu*(xk(1)/m)/(1+(xk(1)/m)) + uk);

        for i = 2:length(xk)
            xk_next(i,1) = xk(i) + cfl*muu*(-(xk(i)/m)/(1+(xk(i)/m)) + (xk(i-1)/m)/(1+(xk(i-1)/m)));
        end

        xk = xk_next;
    end
    WIP_next = 0.1*xk_next;  % WIP(1) = rho(1) * 0.1; actually this is: INT(rho(1),0,0.1)
    OUT = WIP_next;

end
```

For the third control problem, the process description had to be adapted. The resulting code is described below.

```
function [OUT] = Process(k,WIP,uk,blfactor,varargin)

% nonlinear discrete time representation of the manufacturing process
% obtained by discretization of the PDE in time and place
```

```
% parameters
m = max(size(WIP)); % number of machines
xk = m*WIP;
dx = 1/m;
muu = 2; cflcond = 0.5;
dt = cflcond*dx*m/muu; %dt = 0.5/2 = 0.25 hour = 1 sample
cfl = dt/dx;

% Preventive maintenance for one machine
m_pm = 5; % The machine that is subject to preventive maintenance (if m_pm = 0: pm if OFF)
          % The first and the last machine cannot be chosen
if m_pm < 0 | m_pm == 1 | m_pm >= m error('m_pm is not chosen correctly'); end

pm_start = 10;     % Preventive maintenance start (in hours)
pm_freq = 10;      % Preventive maintenance frequency (start every ..th hour)
pm_length = 1;     % Duration of preventive maintenance (in hours)

pm_ks = pm_start/dt;    % PM start (in samples)
pm_kf = pm_freq/dt;     % PM frequency (start every ..th sample)
pm_kl = pm_length/dt;   % Duration of preventive maintenance (in samples)



if nargin == 5
    demand_outflux = muu*(varargin{1})/(1+varargin{1});
    backlog = 0.0;
    for bb = 1 : blfactor

        if m_pm~=0 & mod(k+bb,pm_kf)>=1 & mod(k+bb,pm_kf)<=pm_kl & k+bb>pm_ks
            % Preventive maintenance for the defined machine m_pm
            xk_next(1,1) = xk(1) + cfl*(-muu*(xk(1)/m)/(1+(xk(1)/m)) + uk);
            xk_next(m_pm,1) = xk(m_pm) + cfl*muu*((xk(m_pm-1)/m)/(1+(xk(m_pm-1)/m)));
            xk_next(m_pm+1,1) = xk(m_pm+1) + cfl*muu*(-(xk(m_pm+1)/m)/(1+(xk(m_pm+1)/m)));

            for i = [2:m_pm-1,m_pm+2:length(xk)];
                xk_next(i,1) = xk(i) + cfl*muu*(-(xk(i)/m)/(1+(xk(i)/m))
                               + (xk(i-1)/m)/(1+(xk(i-1)/m)));
            end
        else
            % No preventive maintenance
            xk_next(1,1) = xk(1) + cfl*(-muu*(xk(1)/m)/(1+(xk(1)/m)) + uk);

            for i = 2:length(xk)
                xk_next(i,1) = xk(i) + cfl*muu*(-(xk(i)/m)/(1+(xk(i)/m))
                               + (xk(i-1)/m)/(1+(xk(i-1)/m)));
            end
        end

        actual_outflux = muu*(xk(m)/m)/(1+(xk(m)/m));
        backlog = backlog + dt*(demand_outflux - actual_outflux);
        xk = xk_next;
    end
    OUT = backlog;
else

    for bb = 1 : blfactor

        if m_pm~=0 & mod(k+bb,pm_kf)>=1 & mod(k+bb,pm_kf)<=pm_kl & k+bb>pm_ks
            % Preventive maintenance for the defined machine m_pm
            xk_next(1,1) = xk(1) + cfl*(-muu*(xk(1)/m)/(1+(xk(1)/m)) + uk);
```

```
            xk_next(m_pm,1) = xk(m_pm) + cfl*muu*((xk(m_pm-1)/m)/(1+(xk(m_pm-1)/m)));
            xk_next(m_pm+1,1) = xk(m_pm+1) + cfl*muu*(-(xk(m_pm+1)/m)/(1+(xk(m_pm+1)/m)));

            for i = [2:m_pm-1,m_pm+2:length(xk)];
                xk_next(i,1) = xk(i) + cfl*muu*(-(xk(i)/m)/(1+(xk(i)/m))
                               + (xk(i-1)/m)/(1+(xk(i-1)/m)));
            end
        else
            % No preventive maintenance
            xk_next(1,1) = xk(1) + cfl*(-muu*(xk(1)/m)/(1+(xk(1)/m)) + uk);

            for i = 2:length(xk)
                xk_next(i,1) = xk(i) + cfl*muu*(-(xk(i)/m)/(1+(xk(i)/m))
                               + (xk(i-1)/m)/(1+(xk(i-1)/m)));
            end
        end

        xk = xk_next;
    end
    WIP_next = 0.1*xk_next;  % WIP(1) = rho(1) * 0.1; actually this is: INT(rho(1),0,0.1)
    OUT = WIP_next;

end
```

FUN.m

The goal function for the first control problem as defined in Section 5.2 is described in the following code.

```
function J = FUN(U,FW,QQ,RRa,Y_maxmin,yref,ny,Ck,Ccon,pcdelay,p,m,blfactor,xkold,ukold,d)

y = [];
if pcdelay == 0 % No pc delay

    xk = xkold;
    for i = 1 : m
        xk = Process(xk,U(i),blfactor); % x(k+1) to x(k+m)
        y = [y; Ck*xk+d];               % Input m applied at sample m-1 can for the
    end                                 % first time be seen in x(k+m)
    for i = m+1 : p
        xk = Process(xk,U(m),blfactor); % x(k+m+1) to x(k+p)
        y = [y; Ck*xk+d];               % For m = p, output is x(k+1) to x(k+p) = x(k+m)
                                        % For p > m, output is x(k+1) to x(k+p)
    end

else % Full sample pc delay

    xk = Process(xkold,ukold,blfactor); % x(k+1)
    for i = 1 : m
        xk = Process(xk,U(i),blfactor); % x(k+2) to x(k+m+1)
        y = [y; Ck*xk+d];               % Input m applied at sample m can for the
    end                                 % first time be seen in x(k+m+1)
    for i = m+1 : p
        xk = Process(xk,U(m),blfactor); % x(k+m+2) to x(k+p+1)
        y = [y; Ck*xk+d];               % For m = p, output is x(k+2) to x(k+p+1) = x(k+m+1)
                                        % For p > m, output is x(k+2) to x(k+p+1)
    end
```

```
end

yts = y(ny*(p-1)+1:end);
ets = yts - yref(ny*(p-1)+1:end);

e = y - yref;

J = ets'*FW*ets + e'*QQ*e + [ukold;U]'*RRa*[ukold;U];
```

For the second and third control problem, the goal function was defined as a function of the backlog, which results in the code below.

```
function J = FUN(U,FW,QQ,RRa,Y_maxmin,yref,yrefold,ny,Ck,Ccon,pcdelay,p,m,blfactor,
                 q_constraint,xkold,ukold,d,bold,bbref)

bb = [];
if pcdelay == 0 % No pc delay

    b = bold;
    xk = xkold;
    for i = 1 : m
        b = b + Process(xk,U(i),blfactor,yref(i));    % b(k+1) to b(k+m)
        bb = [bb; b];
        xk = Process(xk,U(i),blfactor);               % x(k+1) to x(k+m)
    end
    for i = m+1 : p
        b = b + Process(xk,U(m),blfactor,yref(i));    % b(k+m+1) to b(k+p)
        bb = [bb; b];
        xk = Process(xk,U(m),blfactor);               % x(k+m+1) to x(k+p)
    end

else % Full sample pc delay

    b = bold + Process(xkold,ukold,blfactor,yrefold); % b(k+1)
    xk = Process(xkold,ukold,blfactor);               % x(k+1)
    for i = 1 : m
        b = b + Process(xk,U(i),blfactor,yref(i));    % b(k+2) to b(k+m+1)
        bb = [bb; b];
        xk = Process(xk,U(i),blfactor);               % x(k+2) to x(k+m+1)
    end
    for i = m+1 : p
        b = b + Process(xk,U(m),blfactor,yref(i));    % b(k+m+2) to b(k+p+1)
        bb = [bb; b];
        xk = Process(xk,U(m),blfactor);               % x(k+m+2) to x(k+p+1)
    end

end

bts = bb(ny*(p-1)+1:end);
ets = bts - bbref(ny*(p-1)+1:end);

e = bb - bbref; % e = bb - bbref  = bb - 0

J = ets'*FW*ets + e'*QQ*e + [ukold;U]'*RRa*[ukold;U];
```

H.3 Description of the nl-MPC implementation

The implementation of nl-MPC consists of eight MATLAB® files. The function of each of these files is described in table below.

MATLAB® files for implementation of nl-MPC	
nlmpc.m	Main file of the implementation. For further detail, see the description below this table.
Process.m	This file describes the internal model of nl-MPC. Here, output values and backlogs are calculated using the (future) inputs.
FUN.m	Definition and calculation of the goal function.
NONLCON.m	This file is used to check whether the nonlinear constraints (if any) are met for the set of (future) inputs that is determined by the optimization algorithm.
Matrices_constant.m	For the optimization algorithm several matrices are required that are constant in time. These matrices are created in this file.
Setpoint.m	Definition of the reference trajectory.
plothorizon.m	With this file, the control- and prediction horizon can be plotted for samples between the start- and end time.
plotresults.m	With this file, the results are plotted once the end time has been reached.

For the files *nlmpc.m*, *Process.m*, and *FUN.m* a short explanation is given on the code as presented in Appendix H.2.

Description of *nlmpc.m*

The file *nlmpc.m* is the main file of the implementation of nl-MPC. Its functions can roughly be split into three parts:

- definition and checking of required input data
- initialization of simulation
- simulation and optimization

These parts are now shortly described. For further explanation, the reader is referred to the comments that are included in the code.

In the first part, all input data that are required for the simulation and application of nl-MPC are collected. These data include: initial state, initial boundary condition,

parameters for nl-MPC and definition of constraints. Subsequently, these input data are checked for errors and the current settings are written to the screen.

The second part is used for initialization of the simulation and optimization. The input data are processed in such a way that suitable matrices for the simulation and optimization result. The file *Matrices_constant* is also used for this initialization.

In the final part of *nlmpc.m*, the simulation is executed. For each sample, an optimal set of inputs is determined by the optimization algorithm *fmincon*. This algorithm makes use of the functions *FUN.m* and *NONLCON.m* that are concerned with respectively the goal function and the nonlinear constraints (if any) of the optimization problem. After this optimization, the first input is applied to the process and all data are updated. At the end of one simulation loop, the control- and prediction horizon as well as the already realized trajectory can be plotted using *plothorizon.m*. Subsequently, the time is increased with one block sample and the optimization is performed again, et cetera. The simulation loop continues until the simulation end time (defined in the first part of this file) is reached. Finally, the results are saved in *testresults.mat* and plotted using *plotresults.m*.

Description of *Process.m*

This file is the internal approximation model of the process that is controlled. It is called by other functions that require either WIP or backlog data. Since these data are closely related to each other, they are both calculated in the same file. It depends on the input that is given to the file whether the WIP or the backlog is returned at the end of the calculations. If the input consists only of 3 terms (WIP, uk, and *blfactor*) and the variable input *varargin* is thus an empty list, the new WIP of the process is returned. If the input however consists of 4 term (WIP,uk, *blfactor*, and *varargin*), then the backlog of the process is returned. In this case, the variable input *varargin* represents the reference value for the WIP at the last workstation in the line.

First the parameters of the process are defined or derived. Secondly, either the new WIP or the backlog is calculated for one block sample (recall that one block sample consists of *blfactor* samples), using the WIP of the previous sample, the input of the current sample and (if necessary) the reference value for the WIP, that are provided in the input data. For this calculation, first the WIP is transformed into density values. Then, the new values of the density are calculated using the fully discretized PDE of Model 3. Finally, either the backlog is determined or the density values are transformed into WIP values again, after which the required output data are returned to the file that requested them.

The file that describes the adapted process for control problem 3 (Section 5.2), is similar to the one described above. The only difference is that for times of preventive maintenance, the outflux of machine 5 (and thus the influx of machine 6) is set at zero. In order

to know when preventive maintenance should take place, an extra input parameter is required: the value of the current sample k.

Description of *FUN.m*

The function *FUN.m* is used by the optimization algorithm *fmincon* in order to calculate the value of the goal function for a certain set of (future) inputs. Since the function is a part of *fmincon*, it gets the same input parameters as *fmincon*. Note that most of these input parameters are not used in this file.

In order to determine the value of the goal function for a certain set of inputs, the output values (for control problem 1) or the backlog values (for control problems 2 and 3) are predicted using *Process.m*. Once the outputs or backlog values are predicted for the whole length of the prediction horizon, they are compared to the reference values. The weighted sum of the deviations between these values is calculated in order to obtain the value of the goal function.

It can be seen in the code of *FUN.m* that other terms appear in the definition of the goal function as well. These terms represent the final state weighting and the input weighting. The final state weighting is not discussed here, since it was not used during the experiments with nl-MPC ($FW = 0$). With the input weighting, the difference between two subsequent inputs is weighted, in order to prevent a fast oscillating input signal. Since this weighting hardly has any effect on the resulting controller, it is not mentioned in the rest of this report.

Finally, the term *pcdelay* appears in the code. This parameter can be used to introduce one sample computational delay, in case the computer time is not negligible to the sample time, but still is smaller than the length of one sample. Since this option is not used in the experiments with nl-MPC ($pcdelay = 0$), it is not discussed further in this report.

Appendix I

Implementation of the control framework

In order to complete the control framework, the nl-MPC controller is coupled to the discrete event model (DEM). For the implementation this means that the files of nl-MPC needs to be connected to the χ-file of the discrete event model. This connection is realized using a Python file [Hof03].

In the implementation of the control framework, the compiled χ file is the main file. With this file, the simulation with the DEM is performed and through functions in the χ code, the MATLAB® files that represent the observer and the nl-MPC controller are called. In the remainder of this appendix, each step of this implementation is discussed. First the χ code is considered. After that, the Python file that links the DEM with the observer and nl-MPC controller is discussed. Finally, the changes in the implementation of nl-MPC come to the attention.

The complete set of files that were used for the implementation of the control work are included on the CD-ROM that goes with this report.

I.1 Description of the χ code

In principle, the χ code is similar to the code used for the validation model. The three processes that have changed are discussed here. These are the exit process E, the generator process G and an additional process $DATA$.

```
proc E(a: ?lot) =
|[ x: lot | *[ a?x ] ]|
```

The exit process has been replaced by this simple process, which only receives finished lots. The processing of output data has been moved to the process $DATA$.

```
proc G(a: ?real, b: !lot) =
|[ i: nat, lambda: real, td: ->real
 | i:= 1
 ; a?lambda
 ; td:= uniform(0.0,1.0)
 ; *[ true; delta -1/lambda*ln(sample td)
       -> b!<time,i>
        ; i:= i + 1
    | true; a?lambda
       -> skip
    ]
]|
```

The generator process has been modified. Instead of the fixed mean arrival rate, now the arrival rate is determined each sample by the nl-MPC controller. Via the process $DATA$, process G receives the mean arrival rate *lambda* that is used during the new sample. Since it would be wrong to create a new distribution for every sample, here only one uniform distribution is used, that is transformed into an exponential distribution using $-1/lambda * ln(sample\ td)$. In this way, the mean of the exponential distribution can be changed every sample without creating a new distribution.

```
proc DATA(a: !input, b: (?nat)^10, c: (!void)^10) =
|[ Tmpc: input#state#dist, Tobs: state#dist, Tend: nat
 , u: input, x: state, d: dist, yp: output
 , t,tmpc: real
 | u:= uinit; x:= xinit; d:= dinit
 ; t:= 0.0; tmpc:= 0.0
 ; *[ true; delta t - time
     -> c.9!; b.9?yp
      ; [ t < tmpc + 0.001 and t > tmpc - 0.001
          -> Tmpc:= MLmpc(u,x,d,yp,time,ts,tobs)
           ; u:= Tmpc.0; a!u
           ; x:= Tmpc.1
           ; d:= Tmpc.2
           ; tmpc:= tmpc + ts
        | t >= tmpc + 0.001 or t <= tmpc - 0.001
          -> Tobs:= MLobs(u,x,d,yp,time,ts,tobs)
           ; x:= Tobs.0
           ; d:= Tobs.1
        ]
      ; t:= t + tobs
    | true; delta tend + 0.001 - time
       -> terminate
    ]
]|
```

Finally, the new process $DATA$ is considered. In this process, for each sample of the observer, the output data of the DEM (i.e., the WIP of the last workstation) is written to the observer by means of the function $MLobs$. With this function, the observer in MATLAB® is called and the new value of the disturbance parameter is returned. Furthermore, the value of the predicted WIP is updated as well. The other input parameters of $MLobs$ are necessary for the calculations in MATLAB®.

At each sample of the nl-MPC controller, the output data of the DEM is written to the nl-MPC controller by means of the function $MLmpc$. The nl-MPC controller returns a

new input value (which is sent to process G) and an update of the disturbance parameter and the predicted WIP. The other input parameters of *MLmpc* are required for the calculations in MATLAB®.

After the simulation end time has been reached, the simulation is terminated.

I.2 Description of the Python code

```
#!usr/bin/python2.2

# chi2ml.py
# Version 1, 12 december 2003
#
from Numeric import *
import pymat

def MLmpc(u,x,d,yp,t,ts,tobs):
    #print "MLmpc function:"
    a = array([u])
    b = array([x])
    c = array([d])
    e = array([yp])
    f = array([t])
    g = array([ts])
    i = array([tobs])

    H = pymat.open("matlab -nosplash")

    pymat.eval(H,"diary on") # Creates log-file
    pymat.eval(H,"clear all")
    pymat.eval(H,"close all")
    pymat.eval(H,"clc")

    pymat.put(H,"ukold",a)
    pymat.put(H,"xkold",b)
    pymat.put(H,"dold",c)
    pymat.put(H,"ypold",e)
    pymat.put(H,"t",f)
    pymat.put(H,"ts",g)
    pymat.put(H,"tobs",i)

    if t==0:
      #print "- mpcinit"
      pymat.eval(H,"mpcinit")

    #print "- nlmpc"
    pymat.eval(H,"nlmpc")

    unew = pymat.get(H,"uk")
    xnew = pymat.get(H,"xk")
    dnew = pymat.get(H,"d")
    unew = unew[0]       # Take element out of 1x1 'matrix'
    xnew = tuple(xnew)
    dnew = dnew[0]       # Take element out of 1x1 'matrix'

    pymat.close(H)
    #print "MLmpc function ended"
```

```
    return (unew,xnew,dnew)

def MLobs(u,x,d,yp,t,ts,tobs):
    #print "MLobs function:"
    a = array([u])
    b = array([x])
    c = array([d])
    e = array([yp])
    f = array([t])
    g = array([ts])
    i = array([tobs])

    H = pymat.open("matlab -nosplash")

    pymat.eval(H,"diary on") # Creates log-file
    pymat.eval(H,"clear all")
    pymat.eval(H,"close all")
    pymat.eval(H,"clc")

    pymat.put(H,"ukold",a)
    pymat.put(H,"xkold",b)
    pymat.put(H,"dold",c)
    pymat.put(H,"ypold",e)
    pymat.put(H,"t",f)
    pymat.put(H,"ts",g)
    pymat.put(H,"tobs",i)

    #print "- observer"
    pymat.eval(H,"observer")

    xnew = pymat.get(H,"xk")
    dnew = pymat.get(H,"d")
    xnew = tuple(xnew)
    dnew = dnew[0]       # Take element out of 1x1 'matrix'

    pymat.close(H)

    #print "MLobs function ended"
    return (xnew,dnew)
```

The functions *MLobs* and *MLmpc* that are defined in this file can be called by the expressions with the same name in the χ code. After one of these functions is called, the input parameters are adapted, MATLAB® is opened, the parameters are imported and the desired function is executed. Afterwards, the results of the function are returned to Python, MATLAB® is closed and the parameters are returned to the discrete event model.

I.3 Changes in the nl-MPC implementation

The nl-MPC implementation as described in Appendix H only requires a few changes before it can be used in the control framework. Since these changes are small, the code is not presented here. The interested reader can find this code on the CD-ROM that goes with this report.

The first change that is required, is the subdivision of the main file *nlmpc.m*. The first and second part of this file (input and initialization) only have to be executed once per simulation. It would therefore be a waste to execute the whole file each time the function *MLmpc* is called. These parts of the file are therefore moved to the file *mpcinit*, which is only performed at the start of a simulation. Furthermore, the observer, which was also a part of *nlmpc.m* is put in a different file, namely *observer.m*. The part that remains in *nlmpc.m* is the optimization part with *fmincon*. Since the optimization should be performed only once each time *MLmpc* is called, the time loop is also removed from the file.

The second change is that the observer is called with a higher frequency than the nl-MPC controller. In fact, this is not really a change in the nl-MPC implementation, since it is determined in the χ code at what times the observer or nl-MPC controller are called.