Event-driven control in theory and practice

Trade-offs in software and control performance

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
Rector Magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op dinsdag 19 december 2006 om 16.00 uur

door

Jacobus Henk Sandee

geboren te Wissenkerke

Dit proefschrift is goedgekeurd door de promotor:

prof.dr.ir. P.P.J. van den Bosch

Copromotor:
dr.ir. W.P.M.H. Heemels

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Sandee, Jacobus H.

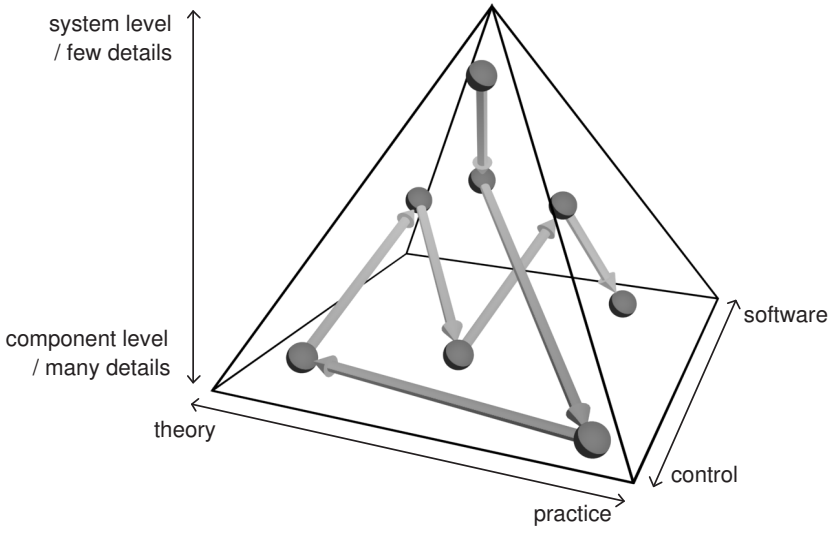
Event-driven control in theory and practice : trade-offs in software and control performance / by Jacobus Henk Sandee. - Eindhoven : Technische Universiteit Eindhoven, 2006.
Proefschrift. - ISBN-10: 90-386-1933-2
ISBN-13: 978-90-386-1933-0
NUR 959
Trefw.: discrete regelsystemen / reactieve computersystemen / bemonsterde regelsystemen / randapparatuur ; printers.
Subject headings: discrete event systems / computerised control / sampled data systems / printing industry.

This PhD-study has been carried out as part of the Boderc project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Dutch Ministry of Economic Affairs under the Senter TS program.

Preface

This PhD-research is part of the Boderc research project: Beyond the Ordinary: Design of Embedded Real-time Control; coordinated by the Embedded Systems Institute. The project aims at developing a model-based methodology to support early decisions over multiple disciplines in the design of high-tech systems. A high speed document printing system of Océ Technologies BV is taken as a case study and acts as the main industrial driver for the project.

According to the Boderc project plan, modern high-tech systems, such as for example encountered in high speed digital printing, rely during the design on the input from different disciplines like electrical engineering, mechanical engineering, software engineering, and control engineering. These disciplines use different models of computation, such as differential equations, automata and finite state machines. Academic research is needed to create models that can effectively bridge the gap between the different disciplines, by mixing different models of computation at the right level of abstraction. However, it is not yet understood how the complete design process can be done under demanding industrial product circumstances. This is the overall topic of the Boderc project. This PhD-thesis specifically focusses on the system design aspects where the disciplines software and control engineering are involved. This is visualized as one of the three axis of a pyramid in the figure below.



To make multi-disciplinary design choices, a system architect typically works with high-level system descriptions, with only few details, as well as with low-level component models, including lots of details. This is illustrated on the vertical axis of the pyramid. To make well-founded trade-offs, he has to be able to reason at these different levels of detail and iterate in short cycles. These different levels can also be observed in this thesis as it presents results at the system level (chapters 1-3 and 7), as well as at the component level (chapters 4-6). The system level results should typically be placed in the emerging field of *systems engineering*, while the lower level results are especially focussed on the domain of *control engineering*, including *software engineering* aspects.

In the multi-disciplinary design of high-tech systems, it is important that academia works closely together with industry. Industry needs the highly specialized, often theoretical knowledge from academia, and academia needs the industrial environment to focus on the *right* problems and to test their ideas within industrial practice and constraints. These aspects can also be recognized in this thesis as the presented concepts are well-founded by academic proofs at several points, as well as validated via practical experiments in the industrial environment of printer development. This is indicated at the third axis of the pyramid. The balls in the pyramid indicate how a typical chapter outline of this thesis moves through the pyramid. This process is also observed in practice when designing high-tech systems.

The pyramid also reflects the nature of the Boderc project: It covers the broad range from system level to mono-disciplinary knowledge on one hand, and from academic to industrial results on the other.

Abstract

Event-driven control in theory and practice
Trade-offs in software and control performance

Feedback control algorithms are indispensable for the proper functioning of many industrial high-tech systems like copiers, wafer steppers and so on. Most research in digital feedback control considers periodic or time-driven control systems, where continuous-time signals are represented by their sampled values at a fixed frequency. In most applications, these digital control algorithms are implemented in a real-time embedded software environment. As a consequence of the time-driven nature of controllers, control engineers pose strong, non-negotiable requirements on the real-time implementations of their algorithms as the required control performance can be guaranteed in this manner. This might lead to non-optimal solutions if the design problem is considered from a broader multi-disciplinary system perspective. As an example, time-driven controllers perform control calculations all the time at a fixed rate, so also when nothing significant has happened in the process. This is clearly an unnecessary waste of resources like processor load and communication bus load and thus not optimal if these aspects are considered as well.

To reduce the severe real-time constraints imposed by the control engineer and the accompanying disadvantages, this thesis proposes to drop the strict requirement of equidistant sampling. This enables the designers to make better balanced multi-disciplinary trade-offs resulting in a better overall system performance and reduced cost price. By not requiring equidistant sampling, one could for instance vary the sample frequency over time and dynamically schedule the control algorithms in order to optimize over processor load. Another option is to perform a control update when new measurement data arrives. In this manner quantization effects and latencies are reduced considerably, which can reduce the required sensor resolution and thus the system cost price. As it is now an event (e.g. the arrival of a new measurement), rather than the elapse of time, that triggers the controller to perform an update, this type of feedback controllers is called *event-driven control*.

In this thesis, we present two different event-driven control structures. The first one is *sensor-based* event-driven control in the sense that the control update is triggered by the arrival of new sensor data. In particular, this control structure is applied to accurately control a motor, based on an (extremely) low resolution encoder. The control design is based on transforming the system equations from the time domain to the angular position (spatial) domain. As controller updates are synchronous with respect to the angular position of the motor, we can apply variations on classical control theory to design and tune the controller. As a result of the transformation, the typical

control measures that we obtain from analysis, are formulated in the spatial domain. For instance, the bandwidth of the controller is not expressed in Hertz (s^{-1}) anymore, but in rad^{-1} and settling time is replaced by settling *distance*. For many high-tech systems these spatial measures directly relate to the *real* performance requirements. Moreover, disturbances are often more easily formulated in terms of angular position than in terms of time, which has clear advantages from a modeling point of view. To validate the theory, the controller is implemented on a high speed document printing system, to accurately control a motor based on an encoder resolution of only 1 pulse per revolution. By means of analysis, simulation and measurements we show that the control performance is similar to the initially proposed industrial controller that is based on a much higher encoder resolution. Moreover, we show that the proposed event-driven controller involves a significant lower processor load and hence outperforms the time-driven controller from a system perspective.

The aim of the second type of event-driven controllers is to reduce the resource utilization for the controller tasks, such as processor load and communication bus load. The main idea is to only update the controller when it is necessary from a control performance point of view. For instance, we propose event-driven controllers that do not update the control value when the tracking/stabilization error is below a certain threshold. By choosing this threshold, a trade-off can be made between control performance and processor load. To get insight in this trade-off, theory is presented to analyze the control performance of these event-driven control loops in terms of ultimate bounds on the tracking/stabilization error. The theory is based on inferring properties (like robust positive invariance, ultimate boundedness and convergence indices) for the event-driven controlled system from discrete-time linear systems and piecewise linear systems. Next to the theoretical analysis, simulations and experiments are carried out on a printer paper path test-setup. It is shown that for the particular application the average processing time, needed to execute the controller tasks, was reduced by a factor 2 without significant degradation of the control performance in comparison to a time-driven implementation. Moreover, we developed a method to accurately predict the processor load for different processing platforms. This method is based on simulation models and micro measurements on the processing platform, such that the processor load can be estimated prior to implementing the controller on the platform.

Next to these contributions in the field of event-driven control, a system engineering technique called “threads of reasoning” is extended and applied to the printer case study to achieve a focus on the right issues and trade-offs in a design.

In summary, two types of event-driven controllers are theoretically analyzed and experimentally validated on a prototype document printing system. The results clearly indicate the potential benefits of event-driven control with respect to the overall system performance and in making trade-offs between control performance, software performance and cost price.

Contents

1

Introduction

Control algorithms are indispensable for the proper functioning of many high-tech applications, such as automobiles, airplanes, home applications (refrigerator, washing machine, vacuum cleaner), industrial plants, and for instance copiers. Within these high-tech systems, a broad variety of controllers can be found. For instance in a copier, where most controllers are implemented to control the velocity of a motor. These motor controllers are mainly found in the paper path where they drive rollers to transport sheets of paper. However, throughout the whole copier, physical parts are moving by the actuation of a motor. For instance in the scanner, where an array of sensors scans the media, or in the finisher, where paper trays are moved to the right position to catch the sheets of paper. Next to controlling motors, controllers are applied for various other purposes in the copier. One example is the temperature control at the location where the image is fused onto the sheet. Also controllers can be found that are not controlling a physical element of the copier, but for instance take care of synchronized timing over the multiple processors in the system.

Most of these control algorithms have in common that they need to be executed on a real-time software processing platform, under strong real-time conditions to guarantee their required control performance. The major reason is that most controller design methods are based on the requirement that the controller sample moments are uniformly distributed over time, i.e. having fixed sample intervals. Stating differently, effects like timing variation (jitter), which are inevitable in most software implementations, are not included in most analysis and synthesis methods. As a consequence, control engineers pose strong, non-negotiable requirements on the real-time implementations of their algorithms. This is illustrated in figure 1.1, which depicts the control algorithm as a package that is thrown over a brick wall to the software department

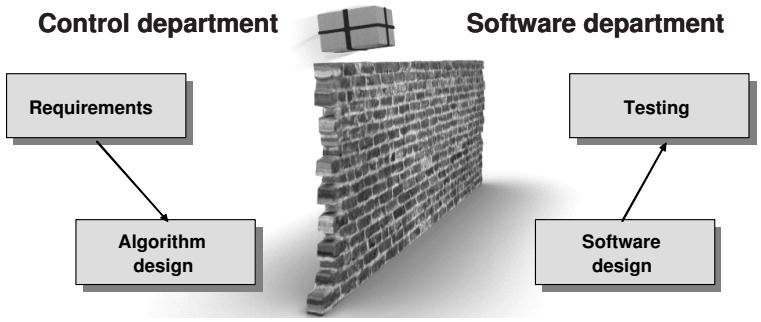


Figure 1.1: "...control engineers pose strong, non-negotiable requirements on the real-time implementations of their algorithms..."

that has to implement and test the algorithm.

An emerging field of research in control engineering is to take the real-time implementation difficulties into account. To be able to guarantee a certain overall system performance, control engineers should be aware of the difficulties in implementing control algorithms with hard real-time requirements in a software environment where resources like processing power and communication bandwidth are shared with other running processes. It is common for them to test their algorithms on dedicated hardware with enough processing power to meet the requirements. They are most often not aware of the underlying scheduling techniques that are used to implement the algorithm on the final product. This means that control engineers typically do not take the needs of the scheduler into account during the design or make use of its properties (e.g. knowledge of deadlines).

But, the reverse is also true: Many (control) software designers have insufficient knowledge of which controller requirements can be relaxed to better fit the underlying software architecture. One example can be found in the choice of the sample frequency. From a control performance point of view, many algorithms are hardly affected by a small variation in the sample frequency, but this might make software implementations a lot easier. Also, control engineers have lots of knowledge about (constant) delays/latencies, that can be handled by the controller, but these are often not communicated to the software designers. Therefore, effective communication is essential, as it is very hard for one person to have extensive knowledge in both disci-

plines. Next to effective communication, however, control design methods should be available to specifically deal with these multi-disciplinary design aspects.

1.1 Trade-offs in software and control performance

There are various issues that make the implementation of controllers difficult on embedded platforms with limited resources. In the previous section we have already mentioned some of them; both from the perspectives of the control engineer as well as from the (control) software designer. These issues result in important trade-offs that affect both the control performance (i.e. tracking, stabilization, disturbance rejection, etc) and the software performance (i.e. processor load, response times, etc) and have to be dealt with in the system design. The main trade-offs are found in:

Sample frequency selection

The sample frequency of the control updates has a strong relation to the required controller bandwidth (the frequency up to which reference signals can be tracked and disturbances are suppressed) [26]. When the sample frequency is increased, the obtainable bandwidth can also be increased by carefully tuning the controller. Often though, the chosen sample frequency is considerably larger than strictly necessary. Typically, the sample frequency is based on rules-of-thumb; for instance at 6 times the desired closed-loop bandwidth when performing discrete-time controller design [25]. In industrial practice the sample frequency might even be chosen at $\geq$40 times the desired closed-loop bandwidth to be at the "safe side".

The reason to lower the sample frequency is obvious in applications where processing power is limited. Each time the control algorithm is executed, a set of tasks needs to be processed, which takes time. The more complex the algorithm, the lower the maximum allowable sample frequency generally is, as it takes more time to execute more complex algorithms (see also 'Time-varying delays' below). Therefore, the main trade-off that can be identified in the selection of the controller sample frequency is between control performance on the one hand and processor load on the other. Studies are carried out to lower the sample frequency requirement by designing the controller in discrete-time, while taking into account the continuous-time behavior of the controlled system

(see e.g. [55]).

Next to the available processing power for the control algorithm, a constraining factor for the sample frequency selection is the limited availability of timer interrupts on the processing platform to trigger the control updates. This is generally the case when more tasks are running on the same processor. It can therefore be favorable to have periodic tasks running on a multiple of each others sample frequency. This indicates some of the demands that software designers might impose on the sample frequency, which are in current practice often not taken into account when designing a controller.

Time-varying delays

Time-varying delays in control implementations are inevitable because of the fact that the execution of tasks on a processor simply takes time. When processing power is limited, often choices have to be made about the complexity of the chosen control algorithm. In many situations, complex control algorithms have the advantage to be able to perform more demanding tasks and achieve a higher control performance. However, when the complexity is increased, the execution time increases, with which the duration until the moment that the actuator signal is updated increases as well. This increased delay generally influences the control performance in a negative way. In practice, this trade-off is taken into account rarely. Note that in the real-time software community one often refers to (fixed) delays as "latencies", and the term "jitter" is used for the time-varying part of the delay.

Other important reasons for (varying) delays caused by the software implementation are: communication of data over a network [86], scheduling (see e.g. [78] in which the effects of various schedulers on the control performance is analyzed), and memory usage (including cache). All these complex issues contribute to the achievable control performance and to the performance of the total system, which makes the controller implementation a difficult multi-disciplinary problem.

The academic literature about control under time-varying delays is expanding rapidly. This topic is particularly of interest as more and more often wireless networks with limited bandwidth are used to communicate sensor and actuator signals over a distance. A nice overview paper of various techniques to analyze the stability of networked control systems is found in [86]. This paper also de-

scribes different types of networks and their influence on the controlled system. In industrial practice, however, control engineers typically only take constant delays into account when designing the controller. The effects of jitter are (at best) quantified in simulations (see e.g. [8]). Another option is to implement the controller with a large but constant delay between measurement and actuation, by implementing a fixed delay that is larger than the maximum expected jitter. In these systems, one often chooses one complete sample period delay.

Quantization

Quantization is often caused by the limited resolution of sensors, but also the software implementation and communication mechanisms can be important reasons. When a controller is implemented on a specific platform we have to deal with a limited resolution for the representation of variables. This is caused by the limited word length. Depending on the processor, calculations will cost more time for bigger word lengths. Therefore, increasing the word length is an advantage for the control performance because of reduced quantization, but might as well be a disadvantage because of the increased computation time. For communicating data over a network with limited capacity, the same reasoning applies.

In control theory, quantization is an effect that is known and has been studied thoroughly since the 1950's. In [25] several methods are discussed how to analyze the control performance of systems affected by quantization. In practice, however, a quantizer is often implemented in simulation to observe its effect on the system behavior. This is an important step, because quantization introduces tracking errors and can cause nasty symptoms (like limit cycles) in the controlled system. Unfortunately, it is also a step that is often overlooked in the early design phases.

Above, we have mentioned various trade-offs between software and control performance in the context of sample frequency selection, time-varying delays and quantization. Still, it lists only a small subset of all the trade-offs that have to be dealt with in the system design, even when only considering the implementation of embedded feedback control algorithms. Moreover, for a high-tech system, multiple disciplines are involved in the trade-off making process. They are all responsible for a specific aspect of the design, e.g. the electronic design, mechanical design and software design, while physics and chemistry constitute clear constraints as well. Designs are often

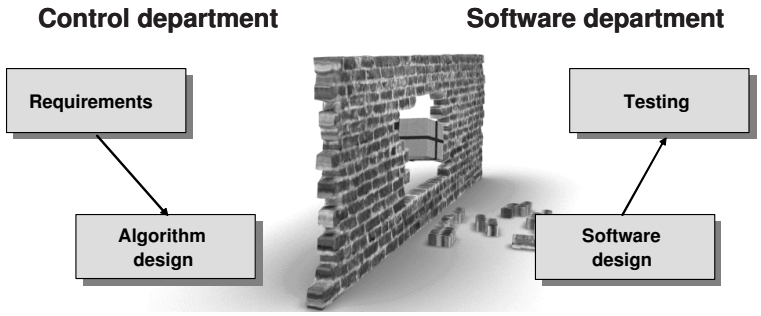


Figure 1.2: "...a step towards breaking down the wall between both disciplines..."

made in parallel by multiple groups of people. Considering the complexity of these systems due to size (one may think typically of millions lines of code and thousands of components like bolts, nuts, etc) and due to many multi-disciplinary design decisions, making these multi-disciplinary trade-offs is a tremendous task and a real challenge. As it is in practice impossible to analyze every trade-off thoroughly due to limited design time and effort, one is forced to zoom in on the most important issues and not waste time on non-issues. Therefore, methods should be available that support in focussing on the most important and most critical issues in a design. This is an essential part in the discipline system engineering [42], as discussed in chapter 2.

1.2 Event-driven control

When considering the trade-offs between software and control engineering, an improvement would be to have design methods for control algorithms that also take the requirements of the software implementation into account. On the other hand, we need design methods for software that deal with the control requirements. Researchers in software and control engineering are becoming increasingly aware of this need for an integrated scientific and technological perspective on the role that computers play in control systems and that control can play in computer systems [75]. The research presented in this thesis focusses on breaking down parts of the wall between both disciplines (figure 1.2), by relaxing one of the most stringent conditions that control engineers impose: a fixed sample frequency.

We propose control algorithms that do not require that sample moments are uniformly distributed over time. We claim that this enables the engineers to make better trade-offs in order to achieve a better overall *system performance*. By not requiring equidistant sampling, one could for instance vary the sample frequency over time and therefore choose to dynamically schedule the control algorithms in order to optimize over processor load. Another option is to design the controller such that it responds faster to acquired measurement data with which quantization effects and latencies are reduced considerably.

These controllers we call *event-driven* controllers, as it is an *event*, rather than the progression of time, that triggers the controller to perform an update. As event-driven control loops typically deal with discrete events having strong interaction with the continuous-time dynamics of the plant, they can be considered as a specific class of hybrid systems. The classical controllers that perform equidistant sampling we call *time-driven*.

Various examples found in literature promote the use of event-driven controllers. For instance, when the event-based sampling is caused by the measurement method used (see e.g. [16, 24, 36, 56, 60]). Also the physical nature of the process that is controlled can be a source for event-based sampling (see e.g. the examples in [21]). When control loops are implemented on embedded platform with control loops closed over networks, it is often very difficult to implement time-driven algorithms, as discussed in [17, 86]. Although we do find examples of event-driven control in literature, hardly any theory on control performance analysis is found. The aim of this thesis is to contribute in filling this gap.

Event-driven controllers better fit the new technology in software and computing platforms, which is focusing on the use of event-based systems. The reason is often to save on computing power and communication load. This is done by adapting existing platforms for event-driven operating systems [20] or by creating new platforms without the need for a system clock [30].

In chapter 3 we will give a more extensive introduction to event-driven controllers and their potential value in practice.

1.3 Experimental and industrial validation

The application context, that is used in this thesis to validate the theoretical results on event-driven control, is best characterized by document printing systems that are

Figure 1.3: The Océ VarioPrint 2090.

highly productive, reliable, and user-friendly. A picture of such a printer is shown in figure 1.3. These systems can print on several sizes of media, different weights, automatically on both sides and include stapling, booklet production, or other types of finishing. In order to be perceived as reliable devices, such printers must be very robust with respect to variations in media, and stochastic variations in timing parameters that relate to paper transport must be controlled up to a high degree. As the printing speed is rather high (typically above 1 image per second), timing requirements are tight and advanced mechatronics are indispensable. This becomes the more apparent if one realizes that the positioning of images on paper has tolerances well below 1 mm.

When considering the embedded control of these systems, one should think of controlling multiple sheets that travel through the paper path simultaneously, while keeping them synchronized with the imaging process. In figure 1.4 a schematic overview of the printer is presented. When the printer is in normal operation, a sheet is separated from the trays in the paper input module, after which it is sent to the paper path that transports the sheets accurately in the direction of the print engine, where the image is fused on a sheet of paper. After that, the sheet is turned for duplex printing, or transported by the paper path to the finisher.

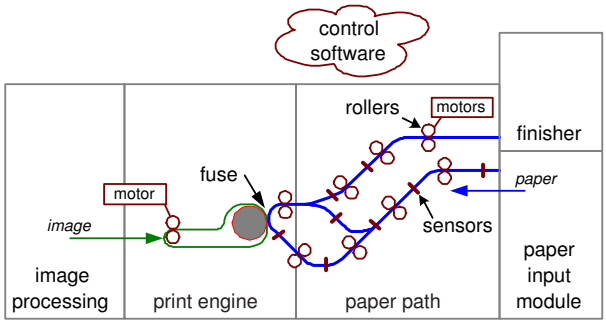


Figure 1.4: Overview of the different components in the printer.

1.4 Scope and outline

Research hypothesis

> Event-driven control improves the overall *system performance* over traditional time-driven control by relaxing one of the most stringent conditions that control engineers impose: a fixed sample frequency.

For the overall *system performance* we consider the system aspects that are affected by the controller's implementation, which particularly are:

- control performance (in terms of tracking, stabilization, disturbance rejection, etc), and
- software performance (in terms of processor load),
- amongst other aspects, like communication bus load and system cost price.

One of the main challenges in this context is to create possibilities to make better balanced trade-offs between the software and control performance. For this, theory to analyze the performance in both areas, as well as practical design rules to implement the event-driven controllers in industry are developed. The results are validated on an industrial high speed document printing system. Next to these contributions in the field of "implementation-aware control", a system engineering technique called "threads of reasoning" is extended and applied to the printer case study to achieve focus on the right issues and trade-offs in the design. Making trade-offs is a fundamental step for any system engineer and this technique is demonstrated to be supportive in this process.

The contributions of the individual chapters are:

Chapter 2: How to focus on the right trade-offs

In the design of technology intensive products (like printers) one searches for a product that satisfies the product requirements as well as the business drivers. The main need in an early design phase is to bring structure in the typical chaos of uncertainty and the huge amount of realization options present. Potential realization choices all have advantages and disadvantages, which cause conflicts in the design. The earlier the essential conflicts are identified, the better it is. Turning them from implicit to explicit helps the system architect in making the trade-off consciously or at least in selecting the most important conflicts that require further in-depth investigation. In this respect we extend the effectiveness of a technique called "threads of reasoning" in the printer case study.

This chapter is based on the work published in the proceedings of the 16th annual international symposium of the International Council On Systems Engineering (INCOSE) [72].

Chapter 3: Event-driven control

This chapter introduces and motivates the use of event-driven control, as opposed to traditional time-driven controllers. The aim of event-driven control is to create a better balance between control performance on one hand, and other system aspects (such as the processor load, communication load, and system cost price) on the other, to achieve a better overall *system performance*. This is done by relaxing the stringent condition of equidistant sampling at high frequencies when applying time-driven control. It is illustrated that event-driven control can reduce the work-load (e.g. processor load, communication bus load) and that response times are reduced as well.

Chapter 4: Sensor-based event-driven control

One of the challenging problems in the design of a printer and many other motion systems is the servo control of several motors at high accuracy. Because of cost price requirements conventional solutions are often not feasible anymore. High resolution encoders are far too expensive and high sample frequencies are prohibitive as controllers have to run on low-cost processors with processing power that is shared with many other tasks. As a possible solution, we present an event-driven controller that is based on an (extremely) low resolution en-

coder. The control value is updated at each moment that an encoder pulse is detected, yielding zero measurement error. Although this approach can be applied in general, we focus on the printer case study, in which we accurately control a DC-motor based on an encoder having a resolution of only 1 pulse per revolution. By means of analysis, simulation and measurements we show that the control performance is similar to the initially proposed industrial controller that is based on a much higher encoder resolution. On top of this, we show that the proposed event-driven controller involves a significant lower processor load, i.e. a better software performance.

This chapter is based on the work submitted for journal publication [73].

Chapter 5: Event-driven control to reduce resource utilization

In spite of the various benefits of using event-driven control, its application in practice is hampered by the lack of a system theory for event-based sampled control systems. The latter is mainly due to the fact that the analysis is mathematically more complicated than for time-driven control loops. To add in building up an event-based system theory, this chapter considers a specific event-driven control scheme for perturbed linear systems. The event-driven control scheme updates the control value only when the (tracking or stabilization) error is large, but not when this error is small. In this way, the average processor and/or communication load can be reduced considerably. The analysis in this chapter is aimed at the control performance in terms of practical stability (ultimate boundedness). By using the derived results, the event-driven controller can be tuned to get satisfactorily transient behavior and desirable ultimate bounds on one hand, while reducing the required resource utilization for its implementation on the other. Several examples illustrate the theory.

This chapter is partially based on the work that appeared in the proceedings of the American Control Conferences 2005 [70] and 2006 [38], and is submitted for journal publication [40].

Chapter 6: Processor load for event-driven controllers

In literature [4, 22, 41, 66], some proposals are made for event-driven controllers to reduce the number of control updates without deteriorating the control performance significantly. The assumption is made that the reduction in control updates results in a reduction of the processor load needed for its implementation.

However, this is unclear as event-driven control typically introduces some overhead and experimental validation of the reduced processor load for event-driven controllers has not been presented in literature so far. This chapter contributes in filling this gap. Simulations, as well as experiments on a printer paper path test setup, show that a reduction in the number of control updates indeed results in a considerable reduction of the processor load, with only a small decrease of control performance. Furthermore, a method is presented to predict the processor load accurately, without having to implement the controller on a test setup.

This chapter is partially based on the work published in the proceedings of the IEEE Conference on Control and Applications 2006 [71] and is submitted for journal publication [74].

Chapter 7: Conclusions and recommendations

Conclusions and recommendations are presented here.

2

How to focus on the right trade-offs[1]

2.1 Introduction

The complexity of products being designed by industry today is increasing at an astonishing rate. The search is for a product that will satisfy the design drivers within certain margins. Design drivers are the important system aspects on which design decisions are based. Examples are: development costs, production costs, time-to-market, throughput, response time, productivity, physical dimensions, power consumption, noise production, and so on. Often, design drivers are conflicting, so that trade-offs must be made.

The main need in the design process of a product is to bring structure in the typical chaos of uncertainty and the huge amount of realization options present. This is most profound in the early design phase. Even typical product requirements might be uncertain in the sense that they are only known up to a certain degree or are still open for discussion. Potential solutions or applied technologies all have advantages as well as disadvantages, which causes *conflicts* in the design. A *conflict* is the situation where a specific design choice influences one or more design drivers positively, while influencing others in a negative way. For instance, in the design of a printer one might consider using stepper motors, DC servo-motors or a combination of both for driving the sheets of paper through the paper path. While stepper motors have the advantage

[1] This chapter is based on the work published in the proceedings of the 16th annual international symposium of the International Council on Systems Engineering (INCOSE) [72].

of being cheaper (particularly as they do not require expensive encoders and because of their long lifetime), they are in general less accurate in positioning the sheets of paper. This causes a conflict between the design drivers *printing accuracy* on the one hand and *cost price* on the other. Of course, more design drivers might play a role in such a decision (e.g. size, power consumption, etc).

This chapter applies the technique of *threads of reasoning* [58] to find such conflicts in the design of the paper flow control in the printer. The technique aims at composing a clear overview of how the conflicts relate to the design drivers. As these relations typically involve multiple design drivers, design choices and their consequences, we refer to these relations as *threads*. The technique is called threads of *reasoning* as the threads typically reveal the *reasoning* applied by the systems engineer. The details of the technique are presented together with a 5-step iterative scheme on how to create the threads. Once the main conflicts are identified *qualitatively*, a further *quantitative* investigation by modeling and measurements is necessary. The specific model-based investigations are only indicated briefly.

In several communities there are alternative and / or related techniques available to identify the main relations and conflicts in the design of a product. For instance, in requirement engineering and more particular in [85] one uses the term "problem bundle" that has similar properties as a thread of reasoning. In [85] these bundles are adopted for structuring a design problem at hand and relating this to the solution space. In product line engineering one has methods like Pulse (see e.g. [9]) and in the system engineering community one uses risk management approaches (see [42, Ch. 6]). These techniques create similar overviews, but more retrospective. Threads of reasoning, on the other hand, is applied throughout the complete system design process and in the various design phases. Therefore, the threads are not static, but continuously changing as the design evolves.

Also in VAP (visual architecting process) (see [57, Ch. 2]) and in ARES (Architectural Reasoning for Embedded Software) [43] related techniques can be found which are especially focussed on software design problems. In TRIZ [3] two important concepts are introduced that are also crucial in our reasoning method: formulating the "ideal" solution to a problem and identifying the conflicts in realizing the ideal product. Quality function deployment (QFD) [65] relates product requirements of the customer to design choices, which from an abstract point of view resembles the reasoning used in this chapter. However, a distinguishing feature of threads of reasoning is that it is graph- instead of matrix-oriented. Matrix-oriented techniques have the tendency

that the number of relationships easily explodes and one easily looses overview of the essential threads. Threads of reasoning is particularly focused on keeping only the essential conflicts, which we consider an advantage. As a consequence, it is possible to graphically represent the overview of the most important design issues. Moreover, most of the mentioned methods have a tendency to move more towards the customer context and less to the realization aspects. The case study here shows how threads of reasoning can also be used to support conceptual and realization choices of the technical design.

The disadvantage of the explosion of the number of relationships is also encountered in a complementary approach in which one archives the design process including the conceptual and realization choices [2]. Often the argumentation why a certain choice has been made is included as well. The documentation typically consists of a chronologically ordered sequence of choices with the aim of traceability: how was a certain choice made at some point in time? If some design changes are made in a later stage, one can still apply the reasoning as kept in the archive. In practice creating and maintaining such an archive is often not feasible due to the enormous complexity. This results in a "tracing" that is not kept up-to-date, with the consequence that its value diminishes. The threads of reasoning technique aims at keeping the essence of the design choices and helps to keep overview.

The outline of this chapter is as follows. In the next section we present the problem statement and put it in the perspective of the multi-disciplinary design of the printer. In section 2.3, the "threads of reasoning" technique is described. In section 2.4, threads of reasoning is applied to identify the most important conflicts in the case study. This leads to conflicts that require a further study via modeling, measurements or other techniques to obtain a well-founded trade-off. In the same section, we indicate briefly which models have been applied to do the in-depth analysis. In section 2.5, the conclusions are stated.

2.2 Problem scope

The problem scope of this chapter is the embedded control design of the paper flow through the printer. The *main* (most important) *design drivers* for this part of the design are:

- throughput (pages per minute),
- printing accuracy (positioning of the image on the sheet),

- time-to-first-print (the time it takes before the first sheet comes out of the printer, after pressing "start")
- power usage,
- cost price and
- time-to-market.

The first three items of this list are typical performance requirements of the printer. Items four and five are constraints on important resources. The last one, time-to-market, is a constraint that is imposed by the organization. The design should be such that all design drivers are satisfied within certain predefined margins.

For the case study considered in this chapter, we assume that the mechanical layout is already given, meaning that positions of (paper transport) rollers, the length and shape of the paper path, etc are known. The design process is in the phase of selecting the control architecture, including:

- Selection of actuators (type and number of motors),
- Selection of sensors,
- Selection of processing architecture (e.g. centralized versus distributed control),
- Selection of operating system (event-driven or periodic architectures?),
- Scheduling of sheets for print jobs.

To support the design process at this stage, the technique of threads of reasoning is applied.

2.3 The technique of threads of reasoning

Threads of reasoning is a graph-based, iterative technique to identify the most important conflicts in the design problem and potential solutions. The system architect uses threads of reasoning implicitly to integrate various views in a consistent and balanced way, in order to design a valuable, usable and feasible product. Architects perform this job by continuously iterating over many different points of view and sampling the problem and solution space to build up an understanding of the case. These threads are *made explicit* by the technique of threads of reasoning.

This technique, as presented in the next section, is based on the work by Muller [58, Ch. 12]. A difference between the technique used here and the one by Muller lies in the used *categories*. The categories are the components of the threads, which

are coupled through their relations. In particular, threads of reasoning in [58] uses the CAFCR framework that adopts the "Customer objectives" (addressing the "what" question from the customer perspective), "Application" (addressing the "how" question of the customer), "Functional" (addressing the "what" question of the product), "Conceptual" and "'Realization" views (addressing the "how" of the product). Instead, it was more suitable in our case to use the following four categories:

- *main design drivers*: limited set of the most important design drivers (typically applying to system level), see section 2.2,
- *sub drivers*: drivers, derived from the main design drivers (typically applying to subsystem level),
- *design choices*: possible solutions or realizations,
- *consequences*: indicating consequences of a design choice.

The threads themselves are formed by multiple connections between the categories above.

2.3.1 Overview of threads of reasoning

Figure 2.1 gives an overview of the iterative process of the threads of reasoning technique. Step 1 is to *select a starting point* for the process. After step 1 the iteration starts with step 2 *create insight*. Step 3 is *deepening the insight* and step 4 is *broadening the insight* via suitable questions. Step 5 *defines and extends the thread*. Moreover, the next iteration is prepared by step 5. In step 5, first the most important and critical threads are selected and one aims at finding conflicts. This insight and refinement might lead to selecting the next need or problem for the new iteration. During this iteration continuous effort is required to *communicate with the stakeholders* (the ones involved in the specific design decisions) to keep them up-to-date, and to *consolidate in simple models* the essence of the problem, and to *update the documentation* to capture the insights obtained.

As mentioned before, the focus of threads of reasoning is to select the critical design issues (step 5) that require in-depth studies to make a sound design trade-off. The in-depth studies are essentially step 3 in figure 2.1. The limited models for *consolidation, communication and reasoning* are derived from these possibly more complex and detailed models for analysis. Especially, since these in-depth studies require a major part of the design time, one has to be selective in the ones that are actually carried

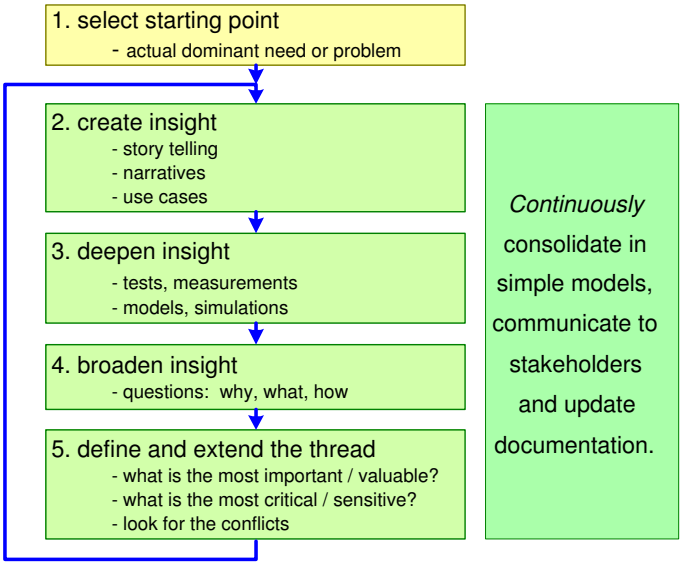


Figure 2.1: Overview of the threads of reasoning approach.

out. Of course, this does not mean that once the answers of these analyses have been obtained, the thread of reasoning is finished. On the contrary, it might actually be altered based on the findings or continued given these new pieces of information.

Below we will describe each of the individual steps in more detail. Moreover, we will present one thread of reasoning as an example from the case study to illustrate the steps.

Step 1: Select a starting point. A good starting point is to take a need or problem that is hot at the moment, within the problem scope. If this issue turns out to be important and critical then it needs to be addressed anyway. If it turns out to be not that important, then the outcome of the first iteration serves to diminish the worries in the organization, enabling it to focus on the really important issues. In practice there are many hot issues that after some iterations turn out to be non-issues. This is often caused by non-rational fears, uncertainty, doubt, rumors, lack of facts, etc. Going through the iteration, which includes fact finding, quickly clarifies the relevance of the issues.

Example. An important issue in the paper flow control is the question how

many processing nodes should be used. Because of the size and the complexity of the software, which is both soft real-time and hard real-time for the various implemented functions, it is almost impossible to process all the code on one node, i.e. one processor. Nevertheless, there are various ways to distribute the software functionality over different (numbers of) nodes. There can be several 'local nodes' that handle separately the control of single motors. Another option is to have only two big processing nodes that handle the entire paper flow control. This design issue is selected as the starting point of the thread.

Step 2: Create insight. In this phase one wants to obtain a rough overview of and insight in the chosen issue. The selected issue can be considered by means of one of the many (sub)methods to create more understanding. Typically, this can be done by the submethods story telling [58, Ch. 11], narratives [19] or scenario-based reasoning using e.g. use-cases [19]. Using these submethods, it will quickly become clear what is known (and can be consolidated and communicated) and what is unknown, and what needs more study and hence forms input for the next step.

Example. To create some first insight into the problem of selecting the number and sizes of the processors in the control architecture, we linked this issue to the main design drivers section 2.2. For the time-to-market to be short, it is important to have a predictable development process. Therefore, a concurrent design process is preferred, which is in favor of having multiple processing nodes. On the other hand, we also want the cost price to be low. Here, the question pops up how the cost price relates to the number of nodes. Looking at the design driver power consumption, there is an obvious relation that more nodes require more power, but more specific information is needed to reveal the exact relation and its importance.

Step 3: Deepening the insight. The insight is deepened by gathering specific facts. This can be done by modeling (and model-based analysis), or by tests and measurements on existing systems. Since the presented technique is iterative, in a first iteration one aims at using simple models, measurements or facts that are obtained in a reasonably short time. Typically, back-of-the-envelope calculations or rules of thumb that are known from previous projects are useful. In a second or subsequent iteration one selects the essential issues (most uncertain, most important) that require more modeling and analysis effort. This

aspect is coupled directly to the Boderc design methodology [39] based on multi-disciplinary modeling: to discover and select the in-depth modeling activities that have to be performed to support the system architect in taking (well-founded) design choices. In the design it is important to only spend time on the crucial issues and not on trivial ones to keep both the design effort and the time-to-market limited. Typically, the models are aimed at shedding light on the conflicts, which were identified earlier (step 5, first iteration).

Example. To get deeper insight in the issues of cost price and power usage of processors, more specific information is needed. A rough quantitative estimate for the cost price showed that a node costs typically about 40 euros, of which 10 euros is calculated for the controller and 30 euros for the printed circuit board (PCB). Because for every node a separate PCB is used, doubling the number of processors roughly means doubling the cost price, although the cost price of the processor can be somewhat less for simple variants. Looking at power demands, it turned out that both the smaller and the bigger processors use about 3 Watt. It would therefore be beneficial to have as few processors as possible. On the other hand, if we look at the power demands from other modules in the printer, that use up to 2 kW, we can assume that the power demand from the processors is of minor importance [27]. Therefore, the power issue will not be included in this thread of reasoning as we aim at describing only the most important aspects.

Step 4: Broadening the insight. Needs and problems are never nicely isolated from the context. Therefore, the insight is broadened by relating the need or problem to the other categories. This can be achieved by answering *why*, *what* and *how* questions. Examples: How can a main design driver be realized by sub drivers? How is a certain issue tackled? Why is a certain design choice good for a specific design driver? What are the consequences of a design choice? How is the consequence related to a specific driver? The insight in the main design driver dimension can also be broadened by looking at the interaction with related system qualities: what happens with safety or reliability when we increase the performance?

Example. What happens if all software would run on two processors? An issue that arises almost immediately from this question are possible synchronization difficulties. This is a typical aspect that needs to be considered in

further iterations. Other example questions for the case-study are: If we separate the software over multiple nodes, how efficiently can the software still be implemented? How would multiple processors be connected?

Step 5: Define and extend the thread. In the previous steps and corresponding discussion of the needs, design choices and problems, many new issues popped up. A single problem can trigger an avalanche of new problems. Key in the approach is not to drown in this infinite ocean full of issues, by addressing the relevant aspects of the problem. This is done by evaluating the following aspects:

1. Which specification and design decisions seem to be the most conflicting?
2. What is the value or the importance of the problem for the customer?
3. How difficult it is to solve the problem? It is important to realize that problems that can be solved in a trivial way should immediately be solved.
4. How critical is the implementation? The implementation can be critical because it is difficult to realize, or because the design is rather sensitive or rather vulnerable (for example, hard real-time systems with processor loads close to 70% or higher, due to which low priority tasks could be blocked for too long).

To evaluate the above aspects, the system architect often uses 'gut-feeling' based on many years of experience. Analysis techniques, such as Failure Mode Effects and Criticality Analysis (FMECA) can be used to analyze the impact of potential problems in the system in a more structured way. Typically, these techniques are used when the design is finished but they can be equally productive during other life-cycle phases of the design process. To compare various solutions, trade studies ([42], section 11.16) can effectively be applied as well.

The next crucial step is to define the thread. In this step the *important* relations between the design drivers, design choices and consequences are represented in a concise diagram. Furthermore, the important conflicts should be clear from the diagram. The problem, that serves as the starting point for the next iteration, can be formulated in terms of this conflict. We believe that a clearly articulated problem is half of the solution.

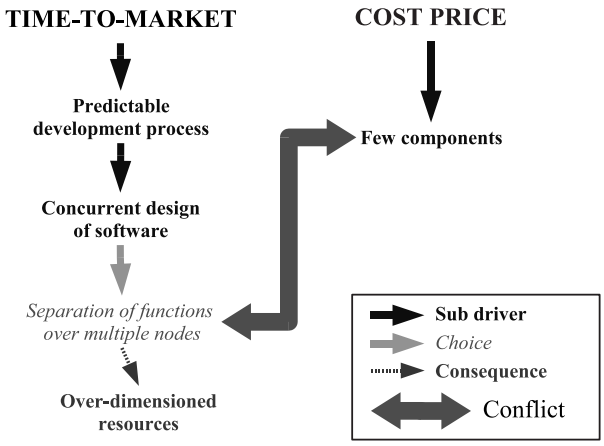


Figure 2.2: Example thread in the design of the paper flow control.

The insights obtained so far, in terms of the most crucial and critical conflicts, should help to select the new need or problem to go into the next iteration (back to step 2).

Example. At this moment in our reasoning on the number and size of processing nodes, the first thread becomes visible, as visualized in figure 2.2. The thread is structured by means of the framework of the categories as introduced before. The interpretation of this visualization is as follows:

- On the top of the picture, the relevant *main design drivers* are given in capitals,
- From the main design drivers, *sub drivers* are derived, indicated in bold face,
- Specific *design choices* that satisfy the sub drivers, indicated in italic,
- The *consequences* that come with specific choices, are depicted with small dashed arrows,
- The main *conflicts*, that are identified between any of the above mentioned aspects of the system, are depicted with thick double arrows.

Note that in step 3 we already concluded that the main design driver power should not be included in this thread. Hence, a step 5 action of discarding less relevant aspects of a thread was already applied. We see that from the question

of how many processing nodes to use, a conflict arises between the drivers 'time-to-market' and 'cost price'. As the most profound conflict is identified now, this can be input for step 2 and subsequently step 3. More detailed models (in comparison with the simple estimates of cost price done earlier) would be useful to deepen the insight, which would support in making the trade-offs in the early design phase. From our first simple models we concluded that for reasons of cost price we want as few processing nodes as possible. However, a proper software design should still be feasible within a limited time span (influencing time-to-market). Therefore, we used a Parallel Object Oriented Specification Language (POOSL) model [62]. With this modeling language and analysis techniques, several possible architectures are evaluated and compared on their feasibility with respect to software timing requirements. Note that a part of the argumentation of a particular choice is captured now in the specific models made. In another setting (or a different architecture) this can be used to reevaluate the design choice. So some kind of "tracing" - as discussed in the introduction of this chapter - is kept.

The thread of reasoning of figure 2.2 was obtained by iterating one-and-a-half times through the 5-step scheme of figure 2.1. As we will see, this is typical for the case at hand as the aim of threads of reasoning in this setting is to select the in-depth models to be made. Normally more iterations - for instance, continuing after the modeling step - are used to find the essential conflicts.

2.4 Threads of reasoning for the case study

The structure that covers the most important threads and their relationships can be complicated for the design of complex systems, like a high-volume document printing system. In addition to the thread presented previously, we will describe two other essential threads in the control of the paper flow. In the presented figures we will use the same interpretation of the visualization as in figure 2.2.

2.4.1 Stepper motors versus DC servo-motors

In this second example thread, the starting point is the use of stepper motors instead of the originally used DC servo-motors for driving the rollers in the printer paper path. The use of DC servo-motors is common for the printer manufacturer and less experience with stepper motors is present.

To create insight (step 2), the use of stepper motors was related to the identified main design drivers. It was easy to see that stepper motors relate to the cost price of the system, as the reason to select them in the first place was the fact that they are cheap. DC servo-motors are more expensive because of their need for (expensive) encoders and shorter lifetime. The use of stepper motors also relates to the printing accuracy. The accuracy of a stepper motor is limited because of various reasons, such as its mechanical construction, cogging and overshoot [28]. Because the stepper motors have to control the movement of the sheet, the sheet can only be controlled with limited accuracy. With a DC servo-motor (in combination with an encoder) the movement of the sheet can be controlled up to much higher accuracy and therefore is no issue.

To see whether the aspects discussed above are really important, we need to deepen our insight (step 3); in this case by quantifying the reasoning. The first aspect was the cost price. The average price of a (low power) stepper motor does not differ that much from the average cost price of a DC-motor. Both can be obtained (for large quantities) for typically less than 10 euros. For both types of motors an electrical driver is required, which also costs about the same for a stepper motor as for a DC-motor, i.e. circa 3 euros for low power applications. An encoder, which is solely needed to control the DC-motor, cannot be obtained below 20 euros for high resolution rotary encoders. This is one of the main reasons why the use of stepper motors is preferred.

Another aspect that needs some quantification is the accuracy of the stepper motor. First measurements reveal that this indeed is an important issue. Figure 2.3 shows a plot of position against time of a stepper motor running at 1 rotation/sec. Four steps are visualized of a 200 steps/revolution motor. The dashed line corresponds to the reference position, the solid line to the actual measured position. The horizontal grid lines indicate the size of the four steps that are visualized. Each step of the motor can be translated to a step-size in the order of 0.2 mm of the paper. From the figure it can be seen that the inaccuracy in the motors position is about 1 step size, i.e. 0.2 mm. As the printing accuracy is defined at 1 mm, the paper needs to be positioned with an accuracy well below 1 mm. The obtained value of 0.2 mm is therefore critical and needs to be evaluated further. It is nevertheless hard to quantify the impact on the real position of the sheet, because of load differences, the occurrence of slip and interactions between two motors that are controlling the same sheet of paper for some period of time. Therefore, more extensive models are needed.

Note that the above reasoning illustrates the typical back-of-the-envelope calcula-

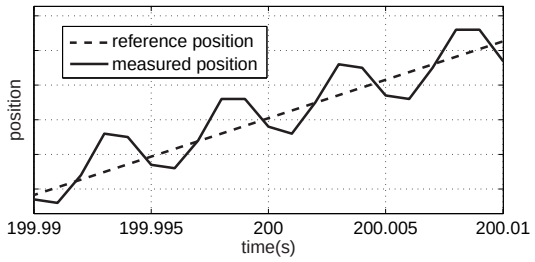


Figure 2.3: Measurement result of stepper motor.

tions that quantify the reasoning.

Like in the first example thread, we broaden our insight by means of the *how*, *what* and *why* questions (step 4). The first question could be how the motor should be controlled. The answer to this question is that a frequency generator needs to be implemented as for every step of the rotor, a drive pulse is needed. The follow-up question to this answer is how this frequency generator could be implemented. This pinpoints the question whether to do this with dedicated hardware or in software. Note that this question is a common struggle in industry nowadays. It comes down to the question whether cost price or accuracy and predictability is more important. Normally, hardware implementations are more reliable and faster or more accurate, but increase the cost price of the system.

The last step in this first iteration is the visualization of the thread. This is depicted in figure 2.4. We see that two important conflicts have been identified that need more attention. The first one is the use of dedicated hardware for the frequency generator in relation to the use of few components to reduce the cost price. The second conflict is identified between the limited accuracy of stepper motors and the requirements on the control accuracy of the sheets.

2.4.2 Time sliced versus event-driven architecture

During the design, a time sliced architecture was proposed for the processing nodes on which, for each node, multiple tasks are scheduled. The idea is that by assigning each task its own time slice, the execution of different functions is temporally separated and task interference is thus avoided. Therefore, software functions can be developed and tested separately while guaranteeing that it will work after combining them on

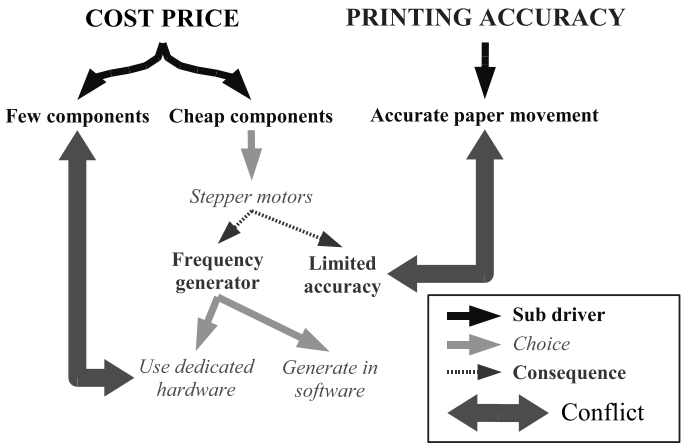


Figure 2.4: Thread of the example of stepper motors.

one processor if each task fits in a slice and there are enough slices. The fact that this choice also has some important disadvantages, makes it a good starting point for a new thread (step 1). To create insight (step 2), we again relate the issue to the main design drivers. The main reason for adapting the time sliced architecture is to shorten the time-to-market, as it enables predictable and composable software design. Furthermore, we can use existing knowledge from past experience of the printer manufacturer (since the time sliced architecture has been applied in the past).

One of the disadvantages of using time slices is the inefficient use of available processing power. Because each task gets a pre-determined part of the available processor time, tasks cannot use the slack time of each other. To quantify the inefficiency of the time sliced scheduling in our case (step 3), we created a simple spread-sheet model which shows the tasks, the expected processor usage and the size of the slices. It also includes an estimation of the interrupts that can occur. Because the interrupts can interrupt any task, a task can effectively take longer to execute than its measured execution time (without interruption). To guarantee the composability of the system, we have to take this interrupt overhead into account for every slice. It turned out that the overhead of the interrupts in a time sliced approach is 20%, while if we replace the time sliced approach by e.g. an earliest deadline first scheduler [79], it becomes much less: 3%.

To broaden our insight (step 4), we could ask ourselves what the influence of the

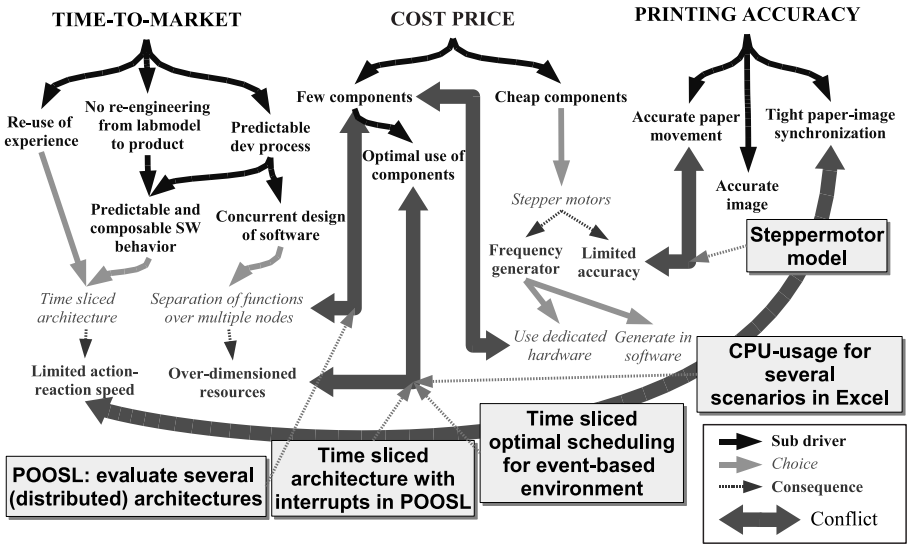


Figure 2.5: Overview of several combined threads of reasoning.

choice of the time sliced architecture would be on the printing accuracy. From past experience, but also from literature it is known that the time sliced architecture introduces a limited action-reaction speed. As we need tight paper-image synchronization for accurate printing, this choice does influence the printing accuracy and therefore needs further in-depth investigation (via modeling).

Figure 2.5 visualizes this thread, together with the first two example threads. From the analysis above, a conflict is identified between the limited action-reaction speed, caused by the time sliced architecture, and the requirement of tight paper-image synchronization.

2.4.3 Total overview

The three example threads are visualized in figure 2.5 in one overview graph. It is interesting to see how these conflicts relate to each other. One example is found in the printing accuracy. The requirement of a high printing accuracy not only conflicts with the use of stepper motors, but also with the use of a time sliced architecture.

With the global overview we have obtained a clear list of conflicts where multi-disciplinary models can be made for deepening the insight (step 3). In figure 2.5, the

light grey boxes are added to indicate the models that have been made. These models give more insight into the identified conflicts. As mentioned before, the threads of reasoning obtained here originate from one-and-a-half cycles through the 5-step scheme to end up with the in-depth models to be made. Although figure 2.5 originates from a limited set of starting issues, related to only a subsystem of the complete printer, and from only one-and-a-half iterations, it already shows a quite complicated structure. Nevertheless, the overview already captures the most important conflicts in the design of the control architecture for the paper path.

2.4.4 Detailed models to obtain insight in conflicts

To deepen the insight, specific models have been made, especially at design considerations where conflicts are identified. Figure 2.5 shows the objects of study of the models in the light gray boxes. To obtain more insight in the conflict explained in section 2.3 (the size and number of processing nodes), a POOSL (Parallel Object Oriented Specification Language) model is created [62]. With this modeling language and the analysis techniques, several possible architectures are evaluated and compared.

A second model was made in the language POOSL to analyze the processor load for the scenario in which the time sliced architecture is 'polluted' with interrupts, which are necessary to make optimal use of components. This is a more detailed model than the spread-sheet model described in section 2.4.2. Both models can also be used to see what the consequences are when the frequency generators for the stepper motors are implemented in software.

To optimally use the processors (and minimize the number of processors), a model was made to calculate optimal schedules for tasks in a time sliced architecture [7]. A stepper motor model, created in Matlab/Simulink, was used to analyze the positioning accuracy of stepper motors [28].

2.5 Conclusions

In this chapter, the technique of threads of reasoning was applied to identify the most important conflicts in the multi-disciplinary design of the paper flow control of the printer. This technique helps to structure in the typical chaos of uncertainty and the huge amount of realization options present in early design phases.

Threads of reasoning is one of the techniques used in the (Boderc) design methodology that aims at using multi-disciplinary models to predict system performance in an

early design phase, while respecting the business constraints of available man power and time-to-market. The restriction in available design time (related to time-to-market and available man power) implies that in-depth and often time-consuming modeling and analysis should be performed only for the essential and critical issues. Threads of reasoning turns out to be - at least in the case of designing the control architecture for a printer - an effective means to find these issues and to create overview.

Combined with the in-depth models, threads of reasoning provides the system architect with valuable insight that supports him in making the important design trade-offs and to reduce some of the uncertainty in the early design phase. It results in a concise picture with the important conflicts depicted *explicitly*. It forces the designer to quantify choices by replacing hand-waving with facts. This stimulates and focuses the discussion with the consequence of a shorter time-to-market and a more predictable design process. Moreover, a part of the argumentation of a particular design choice is captured now in the specific models made and techniques used.

It is a true observation that threads of reasoning itself does not *create* knowledge. It stimulates to make implicit knowledge explicit and indicates where knowledge is lacking and development time should be invested. In this line of reasoning, one could argue whether a technique like this is part of an engineering discipline. This does, nevertheless, not diminish the value of the technique.

Based on the case study, the following suggestions for the use of threads of reasoning can be given:

- Keep the number and the size of the threads limited by selecting the most important ones to keep overview and not to drown in details. In our case study the entanglement was much larger in a first instance of figure 2.5. Additional iterations were used to regain focus and gave rise to figure 2.5 in its present form.

- Whether of not certain conflicts are important, depends, amongst other things, also on the level of the system design. The higher the level, the less detail has to be taken into account. Often though, some iterations will have to go quite deep in a short time to gather some facts that influence design choices at a much higher level. It helps to quantify things (even if the numbers might be uncertain in an early design phase) as it sharpens the discussion and replaces 'gut-feeling' by facts. In particular, back-of-the-envelope calculations, figures-

of-merit and rules-of-thumb help to identify the essential conflicts and to discard the unimportant ones.

- In the reasoning process, fast exploration of the problem and solution space improves the quality of the design decisions. It is important to *sample* specific facts and not to try to be complete. The speed of iteration is much more important than the completeness of the facts. Otherwise the risk is to get stuck within one particular aspect. It is often sufficient to know the order of magnitude and the margin of error for the trade-off analysis (especially in early design phases). Be aware that the iteration will quickly zoom in on the core design problems, which will result in sufficient coverage of the issues anyway.

- It is essential to realize that such an exploration is highly concurrent; it is neither top-down, nor bottom-up. It is typically viewpoint hopping and taking different perspectives all the time.

We applied thread of reasoning to a relatively simple case study, compared to for instance the design of a complete aircraft. To abstract up to more complicated systems, one can apply thread of reasoning recursively on various levels of detail, for the system and subsystem design. In the example of an airplane, one could start with applying threads of reasoning to the overall design, restricting oneself in not taking too much detail into account. Separate threads can then be created of the various decomposed parts of the airplane, such as the motors and the navigation instruments. In the example of the printer, we could have created a separate thread of the image processing and corresponding hardware up to a less detailed thread for the complete system.

An open question still is how to learn the "skill" of threads of reasoning. Being able to iterate fast through the design space and views seems to be hard and tends to be driven by experience. Making the trade-offs in little time seems to be a skill that you can only learn by doing it. However, the guidelines given in this chapter and the presented examples in the case study provide a first step towards learning it.

3

Event-driven control

3.1 Introduction

Most of the research in control theory and engineering considers periodic or time-driven control systems where continuous-time signals are represented by their sampled values at a fixed sample frequency. This leads to equidistant sampling intervals for which the analysis and synthesis problems can be coped with by the vast literature on sampled data systems. The actions, occurring at this fixed frequency, we refer to as *synchronous* in time, as indicated in the first plot of figure 3.1.

The control community typically assumes that the real-time platforms used for implementing controllers are able to guarantee these deterministic sampling intervals. In reality this is, however, seldom achieved, as computation and/or communication delays of networked control systems [46, 61, 86] are inevitable and hard to predict due to the presence of caches and instruction pipelines in modern hardware platforms and the interference between tasks on processors. This typically results in semi-synchronous actions, as shown in the second plot of figure 3.1. By semi-synchronous we mean that actions occur at some average rate, but the sampling intervals vary within certain limits. The delay and jitter introduced by the computer system can degrade the control performance significantly [10]. To study these effects in control loops, in [17] the tools Jitterbug and Truetime are advocated. These tools are mainly based on stochastic analysis and simulations. Other approaches adapt the periodic sampled data theory to incorporate the presence of delay and jitter in servo-loops in the control design. Typically, in this line of work, see e.g. [6, 18, 45, 54, 86], the variations in the "event

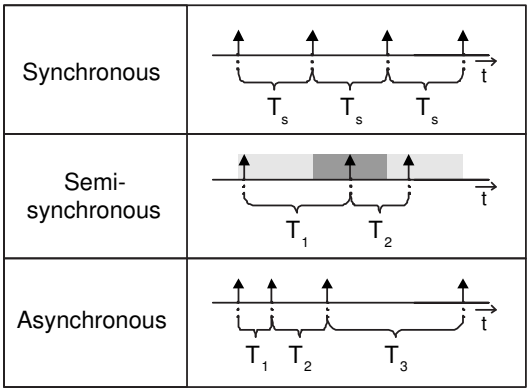


Figure 3.1: Classification of distributions of actions over time: Actions occur *synchronously* in time if the length of the time interval in between each two successive actions is exactly equal. Actions occur *semi-synchronously* in time if there exists a partitioning of the time scale into periods with equal lengths, so that each period contains exactly one action. Actions occur *asynchronously* in time otherwise.

triggering" are considered as disturbances and one designs compensators that are robust to it.

In this research we address the research issue of using event-driven controllers that are not restricted to the synchronous occurrence of the controller actions. We claim that this synchronous sample period is one of the most severe conditions that control engineers impose on the software implementation, which we aim at relaxing (see chapter 1). For event-driven controllers, actions might as well occur asynchronously, as indicated in the third plot of figure 3.1. For these controllers, it is the occurrence of a specific event that triggers the actions to execute. An example can be found in the situation in which the pulse of an encoder is the trigger for the controller to perform an update [36]. When the speed of the motor to which the encoder is connected increases, the rate with which pulses arrive increases as well, which increases the rate at which the controller runs.

For time-driven controllers it is the autonomous progression of time that triggers the execution of actions. To illustrate the difference between time-driven control and event-driven control, take the example of a mailman delivering packages to customers [44]. In the time-driven situation every customer uses the wall clock to check the door for a new package every 5 minutes. When no package has arrived, they can resume their work. In the event-driven situation the mailman rings the doorbell of the specific

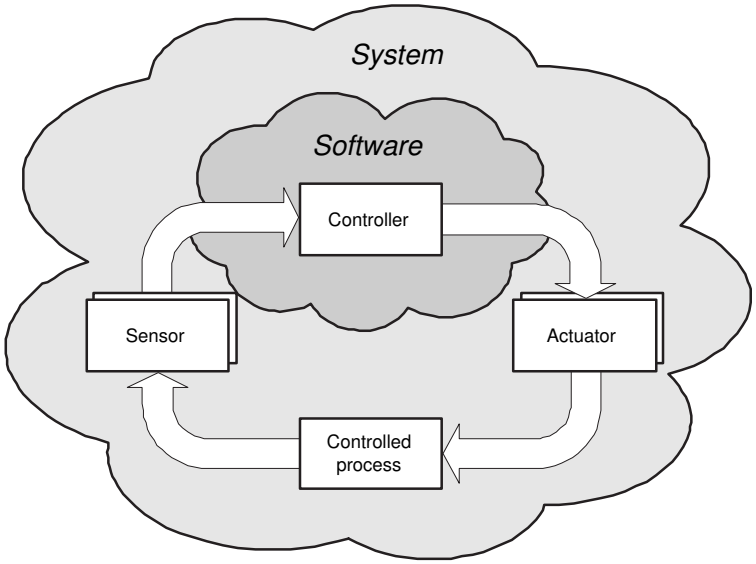


Figure 3.2: Embedded controller scheme.

customer who he has to deliver a package. This customer opens the door and accepts the package. The other customers can continue their work without being interrupted. This example clearly illustrates one of the possible benefits of event-driven control, which is a reduction of the work load, as customers do not have to open their doors unnecessarily. In a control application this is translated to a reduction of, for instance, the communication bus load and processor usage. From a control *performance* point of view, the real advantage of event-driven control is the reduced response times. When a package is delivered, the customer is alerted and can open the door immediately. In the time-driven situation it may take up to 5 minutes after delivery until customers take action to it.

3.2 Examples of event-driven control

Figure 3.2 shows how a controller is normally embedded in a physical system. The controller is connected to one or more actuators to control a specific process. This process is observed by one or more sensors which communicate their observations with the controller. In most cases, the controller is implemented in software. The

controller algorithm is typically just a small part of the complete software, although it is critical in many system performance aspects. It is also possible to have the controller implemented in digital or analogue hardware.

In this thesis, the aim of event-driven control is to create a balance between the control performance and other system aspects. As we will show, event-driven control can for instance reduce the processor load, while maintaining a high control accuracy, by only computing control updates when the measured signal deviates significantly from the reference signal (chapter 5 and 6), or only when new measurement data becomes available (chapter 4). Also sensor resolutions can be reduced considerably, by designing the controller such that it specifically deals with the event-based nature of the sensor (chapter 4). These reduced sensor resolutions have clear cost price advantages.

In literature, only few examples of event-driven control have been presented and hardly any theory on control performance analysis can be found. We believe that in industrial practice event-driven controllers are applied, but because of the lack of theory for these controllers theoretical work is scarce.

An overview of event-driven control examples is found in [4]. Differences in event-driven controllers are related mainly to the event-triggering mechanism used:

- The event-based nature of the sampling can be intrinsic to the measurement method used, for instance, when encoder sensors for measuring the angular position of a motor are used [36]. Other "event-based" sensors include level sensors for measuring the height of fluid in a tank (see e.g. [24, 56]), magnetic/optic disk drives with similar measurement devices [60] and transportation systems where the longitudinal position of a vehicle is only known when certain markers are passed [16]. The event-based nature of these sensors becomes more evident when resolutions are low. This is typically the case when the system cost price becomes important, as high resolution sensors are typically expensive.

- The physical nature of the process being controlled can also be a source for event-based sampling, e.g., in satellite control with thrusters [21], or in systems with pulse frequency modulation [21].

- In modern distributed control systems it is difficult to stick to the time-triggered paradigm. This is specially the case when control loops are closed over computer networks [17, 86] or busses, e.g., field busses, local area networks and wireless networks [46, 61], that introduce varying communication delays.

Next to the various (natural) sources of event-triggering and their relevance in practice, there are many other reasons why the use of event-driven control is of interest:

- As already introduced, a reason why event-driven control is of interest is resource utilization. An embedded controller is typically implemented using a real-time operating system. The available CPU time is shared between the tasks such that it appears as if each task is running independently. Occupying the CPU resource for performing control calculations when nothing significant has happened in the process is clearly an unnecessary waste of resources [4, 5, 22, 41]. The same argument also applies to communication resources. In the case the number of control updates can be reduced this leads directly to a reduction in the number of messages to be transmitted and thus the bus load. As communication busses have limited bandwidth, reducing the bus load is beneficial for the total system. Moreover, when wireless communication is used, lower bus loads also save energy. Especially for battery-powered devices, this is an important aspect as wireless communication is a severe power consumer. Lots of research is carried out in the reduction of power usage for battery-powered devices like wireless sensors [23, 82].

- Also from a technology point of view, event-driven controllers are becoming increasingly commonplace, particularly for distributed real-time sensing and control. For example, for sensors in sensor networks, TinyOS, an event-based operating system, is rapidly emerging as the operating system of choice [53]. In software design, one often uses event-driven algorithms rather than time-driven. These are then scheduled using a priority-based scheduler. In [49] a comparison is presented between the event-driven and the time-driven approach applied in responsive computer systems. This comparison focusses on the temporal properties and considers the issues of predictability, testability, resource utilization, extensibility, and assumption coverage.

- As stated in [4], event-driven control is closer in nature to the way a human behaves as a controller. Indeed, when a human performs manual control his behavior is event-driven rather than time-driven. It is not until the measurement signal has deviated sufficiently enough from the desired setpoint that a new control action is taken [59].

Although it seems in many situations logical to study and implement event-driven

controllers, their application is scarce in both industry and academia. A major reason why time-driven control still dominates is the difficulty involved in developing a system theory for event-based systems. This is mainly due to the fact that the analysis is mathematically more complicated than for time-driven control systems. To add in building up an event-based system theory, we consider two specific event-driven control schemes. The practical value of these controllers is validated in the printer case study.

3.3 Event-driven control in the case study

In the printer case study many controllers can be found. As already mentioned in the introduction, most of these controllers are implemented to control the position or velocity of motors. These motors can be found at various places in the printer, like in the paper path where they are used to drive the rollers that transport the sheets of paper, or in the finisher, where paper trays are moved to the right position to catch the sheets of paper. Next to controlling motors, controllers are applied for various other purposes in the printer. For instance the control of the temperature at the location where the image is fused onto the sheet. Also controllers can be found that are not controlling a physical element of the printer, but for instance taking care of synchronized timing over the multiple processors in the system.

For the case study used in this thesis, we specifically study the motor controllers. Nevertheless, most of the theory and methods presented are not restricted to this type of application, but can be applied in a broader context. In particular, we will study the motor controllers applied at two locations in the printer. The first one, discussed in chapter 4, deals with event-driven control applied to control the motion of the motor that drives the image belt, based on a low resolution encoder. This belt transports the image from the location where the image is created from toner particles to the point where the image is fused onto the sheet of paper (figure 3.3). Secondly, we study a specific type of event-driven controller to control the motors that drive the sheets of paper through the paper path, with the aim to reduce processor load in comparison to traditional time-driven approaches. This is presented in chapters 5 and 6.

Conventionally, all controllers in the printer were time-driven. However, as cost price is becoming more important in the printing industry, system requirements are changing and conventional components and methods do not satisfy anymore. How this affects the motor controllers is visualized in figure 3.4, by means of a threads of

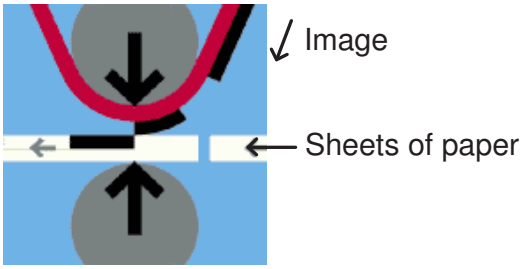


Figure 3.3: Fusing images on sheets of paper.

reasoning diagram (as explained in chapter 2). In the figure it is shown that the reasons to choose either an event-driven controller or a time-driven controller originate from the main design drivers *cost price*, *printing accuracy* and *time-to-market*. To reduce the cost price, the choice was not to apply the conventional high resolution encoders anymore, because they are far too expensive. As time-driven controllers assume exact measurements at synchronous controller sample moments, they cannot easily cope with low resolution encoders that only supply position information at certain discrete angular positions, i.e. at asynchronous time instants. An event-driven controller, however, can use the *exact* position measurement at the moment a pulse is detected.

For cost price reasons one also wants to make optimal use of resources, like processing power, communication bandwidth and electrical power. Time-driven controllers, however, operate with a constant load for resources like processing power, also at moments when nothing significant has happened in the system. For this reason, one would like to design event-driven controllers with minimum (average) processor load, that are only active when really necessary. (think of the mailman example in the introduction of this chapter).

The printing accuracy depends crucially on the performance of the motor controllers that control the motion of the sheets of paper as well as the motion of the image through the printer. To obtain a high control performance, small response times for the control software are necessary to react fast on changing conditions. For these small response times one should choose high sample frequencies when applying time-driven control. This, however, is in contradiction with the minimization of processor load, as high sample frequencies result in high processor loads. Also in combination with low resolution encoders, time-driven control might likely not give the required

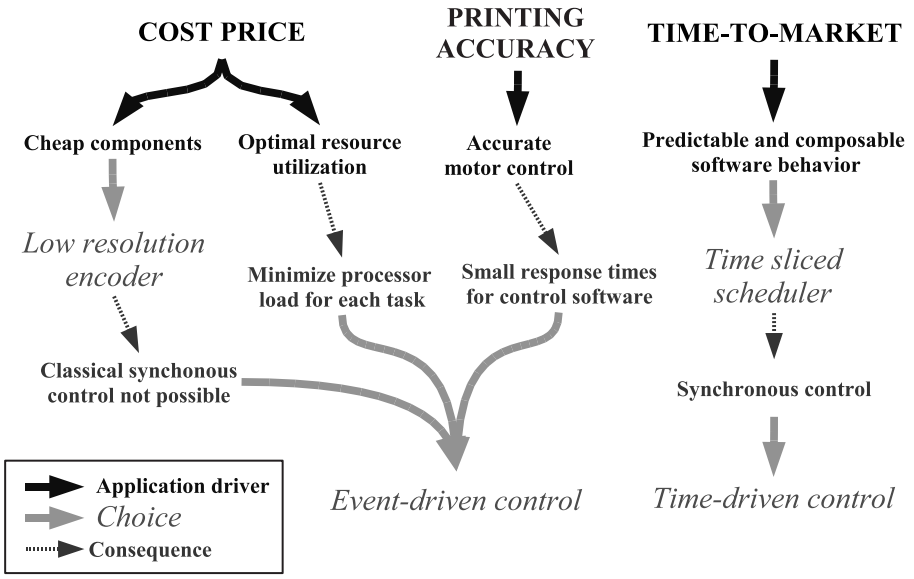


Figure 3.4: Threads of reasoning diagram for event-driven control in printer case study.

control performance. As event-driven controllers are not bound to a constant sample frequency, one can design and implement the event-driven controller such that it responds fast to these changing conditions.

The main advocate of time-driven control is related to the chosen time sliced architecture and thus to the main design driver time-to-market, as explained in section 2.4.2. As the time sliced architecture is designed to schedule tasks with a fixed period time, it is not efficient to implement an asynchronous event-driven controller.

3.4 Contributions in event-driven control

In this thesis, two types of event-driven control schemes are presented, that have been successfully applied in the printer case study. The first one, presented in chapter 4, uses an (extremely) low resolution encoder to measure the angular position of a motor. The event-driven controller is designed such that actuation is performed right after the detection of an encoder pulse. In this way, the controller can use the *exact* position measurement, and is not affected by the quantization errors of the encoder. Moreover, the controller can respond fast to measurement data. When the motor is not running

at constant velocity, the updates are not equidistant in time. It is therefore not possible to use the classical design methods which all assume that updates are equally spaced in time. We can however apply variations on classical design methods if we define our models of the plant and the controller in the (angular) position domain instead of the time domain, as proposed in [36]. This idea is based on the observation that the encoder pulses arrive equally spaced in the position domain. It is shown in chapter 4 that, by applying this event-driven controller, we not only decrease the encoder resolution - and therefore the system cost price - but also the average processor load, compared to the conventional controller. This was accomplished without degrading the control performance, with respect to the originally applied controller.

The focus of the second type of controller, presented in chapters 5 and 6, is to obtain a high control performance on one hand and realizing a reduction of the resource utilization (processor load, communication bus load) on the other. This is realized by updating the controller only when the (tracking or stabilization) error is larger than a threshold and holding the control value if the error is small. Already in 1962, the need for such controllers was addressed [22], but few research has been spent in this subject since. The research presented in chapter 5 aims particularly at a mathematical analysis of such controllers to start building an event-based system theory. The proposed controller is furthermore experimentally validated to research the real benefit in terms of processor load reduction and not only the number of control updates. This is presented in chapter 6.

Hence, the main drivers for event-driven control in the printer case study, as presented in figure 3.4, are effectuated.

4

Sensor-based event-driven control[1]

4.1 Introduction

In industry we observe an ever increasing search for better performing products at decreasing cost prices. Especially for consumer products that are sold in large quantities, like dvd-players and televisions, the price must be low to compete in the tough market. Although cost price should decrease, the requirements rise. In the example of dvd-players, consumers want both more functionality and higher recording speeds and quality.

In the document printing industry the same trend is observed. Printers should operate at higher printing speeds, be able to handle more media simultaneously and produce more accurate prints for the same or even a lower cost price. Next to that, the demands for power consumption and machine size are tightened. Because of these challenging requirements and the need to reduce the cost price, trade-offs appear for many aspects in the product design and system designers are forced to come up with creative solutions for these hard problems.

One of these challenging problems in the design of a printer, but also in many other high-tech systems, is the servo control of several motors at high accuracy. Because of cost price requirements conventional solutions are often not feasible anymore. High resolution encoders are too expensive and high sample frequencies are also prohibitive

[1] This chapter is based on the work submitted for journal publication [73].

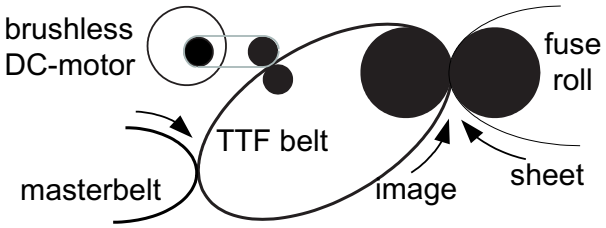


Figure 4.1: Schematic representation of the 'Copy Press system'.

as controllers have to run on low-cost processors with processing power that is shared with many other tasks.

One of the leading companies in high volume document printing systems is Océ Technologies BV. The technology that this company has developed for high speed laser printing is the 'Copy Press system' (see figure 4.1). In this technique the toner of the image is transported from the masterbelt to the fuse roll (where the image is fused onto the sheet) with the Toner Transfer Fusing (TTF) technology. This technology uses a special belt (TTF belt) that transports the image. This TTF belt is accurately controlled by a brushless DC-motor. Accurate control is important as the positioning accuracy of the image directly influences the printing quality.

The brushless DC-motor includes three Hall sensors (sensors that can detect magnets connected to the rotor, and therefore the position of the rotor) to implement correct commutation, as no brushes are available, like in the more commonly used DC-motors. Because of the omission of brushes, brushless DC-motors have a large lifetime and high reliability. In controlling the motor, conventional control algorithms use expensive, high resolution encoders to accurately measure position. For instance, in the conventional printer setup, typical encoder resolutions are 500 pulses per revolution (PPR), with controllers running at 500 Hz. By using these high resolution encoders, measurement quantization errors can be neglected.

To keep the system cost price limited, our aim is to use only the Hall sensors to control the motor. The obtainable resolution is thereby limited to 12 PPR in the practical case-study (after demodulation). However, now the quantization errors become significant and are not negligible anymore. To still achieve satisfactory control performance, this requires an adaption to conventional control algorithms to deal with this low-resolution encoder signal.

Most applied and researched solutions that deal with noisy and low resolution sensor data use an observer-based approach to estimate the data at *synchronous* controller sample moments, based on *asynchronous* measurement moments [11, 16, 29, 31, 50, 60, 67, 84]. In these solutions, the continuous-time plant is translated into a discrete-time model which is time-varying, as it depends on the time between successive measurement instants. The probably most applied example in the field of automatic control is the Kalman filter [33]. In Kalman filtering, the estimate of the system state is recursively updated by processing a succession of measurements. Each controller sample moment, the measurement data is compared with a model-based estimate of the measurement. The difference is used to correct the estimate of the state. To deal with asynchronous measurements, the authors of [50] use an oversampled Kalman filter between two controller time instants. The computation rate of the filter however remains equal to the controller sample frequency, as the Kalman filter equations can be evaluated in batches at the controller sampling instants. [67] presents an adapted distributed Kalman filter to deal with asynchronously operating sensors. The paper presents how the Kalman filter equations can be adapted to deal explicitly with the time a measurement has been taken, not necessarily being a controller sample moment. In [31] a similar structure is presented to perform velocity estimation from irregular noisy position measurements. In the paper, the method is compared to conventional methods based on pulse counting and a method based on extrapolation over the last measurements.

Next to the Kalman-based approaches, several other approaches are presented in literature to estimate the data at synchronous controller sample moments, based on asynchronous measurements. For instance in [60], a Luenberger-type observer is used to use asynchronous measurement data in combination with a multi-rate controller scheme. In tracking applications, a well-known technique is the $\alpha\beta$-tracker [11, 29] to estimate position, velocity and acceleration in a time-discrete manner, often based on extrapolating and filtering measurements. Other methods include backward difference, Taylor series expansion and polynomial fitting through a number of past measurements (e.g. least-squares). An overview of these methods and a comparison between them is presented in [84] in the application of using optical incremental encoders to measure position and velocity.

All above discussed methods have proven to achieve a better control performance compared to the situation in which one neglects the quantization and sensor noise in the measurements. However, the main drawback of these methods is that they

generally require a high computational effort for computing the observer estimates. Furthermore, next to the control parameters, also the observer parameters need to be tuned to get good overall control performance.

A completely different approach has been taken in [36], in which a simple control structure is presented for the control of a slave motor in a master-slave combination, that does not suffer from the added complexity of an observer. The control structure is an asynchronous control scheme in which the control updates are triggered by the slave position measurement (encoder pulse). The idea of the asynchronous controller is based upon the observation that at an encoder pulse the position is *exactly* known and thus there is no need for an observer as in the before mentioned approaches. However, as the velocity of the motors vary over time, both measurement and control updates are not equidistant in time. This requires a completely new design strategy for these event-driven controllers of which initial proposals are made in [36]. The applied design techniques form the basis of the current work.

This chapter applies a similar controller structure as proposed in [36] and extends the controller analysis and design techniques to accurately control the brushless DC-motor in the printer on the basis of a very low resolution Hall encoder. The method to design and tune the controller is presented. A parameter-varying / gain scheduled controller is proposed for the case wherein a printer is able to print at multiple speeds. Furthermore, it is explained how typical control measures, like bandwidth, sensitivity plots, settling behavior, etc, can be derived for the event-driven controller. Simulation and experimental results are compared with an $\alpha\beta$-tracker approach, as originally applied to control the motion of the TTF belt. This latter approach we call the observer-based controller. For the experiments we used the industrial setting of a newly developed high speed printer.

The outline of this chapter is as follows: First a problem statement is given. Next, sections 4.3 and 4.4 present two solutions for the problem: The conventional industrial solution in the form of an observer-based controller based on the $\alpha\beta$-tracker approach, and an event-driven controller, respectively. The design methods for the event-driven controller are presented in section 4.4 as well. Section 4.5 discusses the main implementation issues. Simulation and measurement results are presented in sections 4.6 and 4.7, respectively. Section 4.8 presents results on the implementation trade-off between control performance and processor load. The chapter is concluded with discussion and conclusions.

4.2 Problem formulation

The brushless DC-motor that is driving the TTF-belt (figure 4.1) is modeled by the second-order model

$$\begin{aligned} \dot{\theta}(t) &= \omega(t) \\ \dot{\omega}(t) &= \tfrac{1}{J_m}[(\tfrac{-k^2}{R} - B)\omega(t) + \tfrac{k}{R}u(t) - d(t)] \end{aligned} \tag{4.1}$$

where $\theta(t)$ $[rad]$ is the angular position of the motor axis, ω(t) is its angular velocity $[rad/s]$, $u(t)$ the motor voltage $[V]$ and $d(t)$ the disturbance torque $[Nm]$ at time $t \in \mathbb{R}$. The motor parameters are obtained from data sheets of the motor manufacturer: the motor inertia $J_m = 0.83 \cdot 10^{-4}\ kgm^2$, the motor torque constant $k = 0.028$ Nm/A, the motor resistance $R = 1.0\ \Omega$ and the motor damping $B = 3.0 \cdot 10^{-5}$ Nms/rad.

To the motor axis, several loads are coupled via the TTF belt. The total load is modeled as an inertia $J_l = 1.0 \cdot 10^{-4}\ kgm^2$. For frequencies of input signals and disturbances well below the lowest resonance frequency of the real system, we can model the load by combining the motor inertia and the load inertia: $J = J_m + J_l = 1.83 \cdot 10^{-4}\ kgm^2$. For higher frequencies we could use a fourth order model, derived with techniques as described in [26]. In this research we will use the second order model as given in (4.2), as the required control bandwidth (of 4.0 Hz, as explained below) is well below the lowest resonance frequency (which is located at approximately 10 Hz).

$$\begin{aligned} \dot{\theta}(t) &= \omega(t) \\ \dot{\omega}(t) &= \tfrac{1}{J}[(\tfrac{-k^2}{R} - B)\omega(t) + \tfrac{k}{R}u(t) - d(t)] \end{aligned} \tag{4.2}$$

The industrial requirements for throughput and printing accuracy in the printer result in a feedback and feed-forward controller combination such that:

- The deviation from the steady-state position error is at most 0.25 rad during printing. Only deviations from a constant position error will be visible in the print quality.
- Position profiles are tracked corresponding to constant velocities ranging from 200 rad/s to 500 rad/s.
- Disturbances are rejected sufficiently up to frequencies of at least 4.0 Hz. This is defined as the required controller bandwidth. Below the derivation of this bandwidth is further explained.

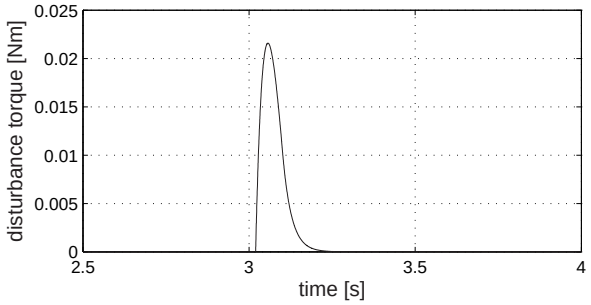


Figure 4.2: Disturbance torque pulse at the motor axis, induced by a sheet entering the fuse roll.

The main disturbances are caused by sheets that enter the fuse roll. The main component of this disturbance is due to the torque needed to open the fuse roll such that the sheet can enter. From measurements at the motor axis that drives the TTF-belt, the disturbance signal as induced by one sheet entrance in the fuse was obtained, as depicted in figure 4.2.

The requirement on the controller bandwidth is derived from the disturbance with the highest frequency that the controller needs to compensate. This disturbance is caused by the roll that is coupled to the motor and drives the TTF belt. As the gear ratio between motor and roll is 20:1, the main component of this disturbance has a period of $20 \cdot 2\pi$ rad at the motor axes. Because the disturbance is varying with the angular position of the motor, the frequency content of this signal varies with the velocity at which the motor is operated. At maximum printing speed of 100 pages per minute, which corresponds to a motor velocity of 500 rad/s, a disturbance at $\frac{500}{20 \cdot 2\pi} = 4.0$ Hz is observed. This disturbance can be modeled as a disturbing torque. From this the requirement was derived that the controller has to sufficiently suppress disturbances up to at least 4.0 Hz.

To measure the angular position of the motor axis, the Hall pulses are used as an encoder with a resolution of 12 PPR. This results in a position update every 0.52 rad of the motor axis, which is equal to an image displacement of about 0.4 mm. It is important to note that the Hall sensors are positioned along the motor axes with an inaccuracy of ± 0.2 rad (which is equal to 3% of one revolution).

Of course, there are some limiting factors in the controller design, such as input saturation, which will be discussed in section 4.5.

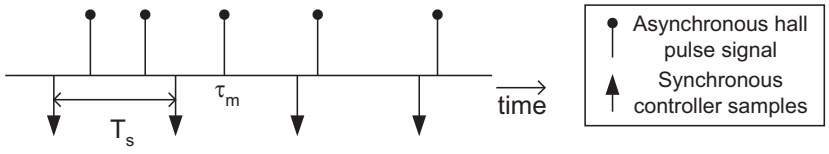


Figure 4.3: Typical pulse sequence for observer-based controller.

4.3 Observer-based control

In the observer-based control scheme, as originally implemented to control the motion of the TTF-belt, the actuator signal is updated at a constant rate with sample period T_s (i.e. synchronous in time) but measurements are done asynchronously in time as depicted in figure 4.3. Each moment a new Hall pulse is detected, a time-stamp (τ_m) is taken. At each synchronous control update, this time-stamp is used to estimate ω and θ at the control update times (kT_s) from the asynchronous measurements. The estimates are denoted by ω_{est} and θ_{est}, respectively. This is done in two steps:

- Based on $\omega_{est}((k-1)T_s)$ (computed at the previous control update time $(k-1)T_s$) the latest measured angular position $\theta_m(\tau_m)$ is translated into an extrapolated angular position (θ_{extr}) at the next synchronous control update time kT_s. This is done by linear extrapolation at the discrete-time instants $k = 0, 1, 2, ...$:

$$\theta_{extr}(kT_s) = \theta_m(\tau_m) + (kT_s - \tau_m)\omega_{est}((k-1)T_s), \tag{4.3}$$

  where $(k-1)T_s \leq \tau_m < kT_s$.

- To deal with high frequency measurement noise, caused by the inaccurate positioning of the Hall sensors, the $\alpha\beta - tracker$ structure [29] was adopted to compute $\theta_{est}(kT_s)$ from $\theta_{extr}(kT_s)$. At the same time, the $\alpha\beta - tracker$ also computes $\omega_{est}(kT_s)$. The state-space description of the $\alpha\beta - tracker$ is as follows:

$$\begin{aligned}\begin{bmatrix} \theta_{est}(kT_s) \\ \omega_{est}(kT_s) \end{bmatrix} &= \begin{bmatrix} 1-\alpha & (1-\alpha)T_s \\ \frac{-\beta}{T_s} & 1-\beta \end{bmatrix} \begin{bmatrix} \theta_{est}((k-1)T_s) \\ \omega_{est}((k-1)T_s) \end{bmatrix} \\ &+ \begin{bmatrix} \alpha \\ \frac{\beta}{T_s} \end{bmatrix} \theta_{extr}(kT_s)\end{aligned} \tag{4.4}$$

  This can also be written as a weighted sum of the estimated values, based on

information at time kT_s, and the "innovation terms", as derived from the extrapolated measurements:

$$\begin{aligned}\theta_{est}(kT_s) &= (1-\alpha)\{\theta_{est}((k-1)T_s) + T_s\omega_{est}((k-1)T_s)\} + \alpha\theta_{extr}(kT_s)\\ \omega_{est}(kT_s) &= (1-\beta)\omega_{est}((k-1)T_s) + \beta\frac{\theta_{extr}(kT_s)-\theta_{est}((k-1)T_s)}{T_s}\end{aligned} \tag{4.5}$$

Based on the difference between the reference values for θ and ω (denoted by θ_r and ω_r, respectively) the error signals can be calculated that enter the controller:

$$\begin{aligned}e_\theta(kT_s) &= \theta_r(kT_s) - \theta_{est}(kT_s)\\ e_\omega(kT_s) &= \omega_r(kT_s) - \omega_{est}(kT_s)\end{aligned} \tag{4.6}$$

To control the motor, a PD controller with feed-forward was used with the following structure:

$$u(t) = K_p e_\theta(kT_s) + K_d e_\omega(kT_s) + K_{\text{ff}}\omega_r(kT_s) \tag{4.7}$$

for $kT_s \leq t < (k+1)T_s$ (i.e. using zero-order hold). In this controller the differential component is in fact calculated from the $\alpha\beta - tracker$ that acts as a differentiator and low-pass filter (by choosing both α and β smaller that 1). The last term in the above equation is the velocity feed-forward that is added to the control output with static feed-forward gain K_{ff}. As only deviations from the steady-state position error, for constant velocity reference tracking, are important for the print quality, we do not need an integral term in the controller.

When tuning the controller we need to find values for $T_s, K_p, K_d, K_{\text{ff}}$, α and β. The sample frequency ($\frac{1}{T_s}$) of the observer-based controller for the existing implementation in the printer was chosen at 250 Hz, which is about 50 times the required bandwidth of 4 Hz. The sample frequency was chosen this high since all controller tuning was done in continuous-time. Discretization by approximating the continuous-time controller was used to implement the controller in discrete-time. This approximation requires the high sample frequency. We will use this same sample frequency in combination with the observer-based controller (4.3)-(4.7) as a reference for comparison with the event-driven controller, presented in section 4.4. In section 4.6 we will investigate how this observer-based controller performs for a much lower sample frequency of only 62 Hz. This will require some re-tuning of the controller.

The values for the PD controller were chosen at $K_p = 2$, $K_d = 0.3$, $\alpha = 0.75$ and $\beta = 0.25$ such that the relative damping of the system is larger than 0.45 and

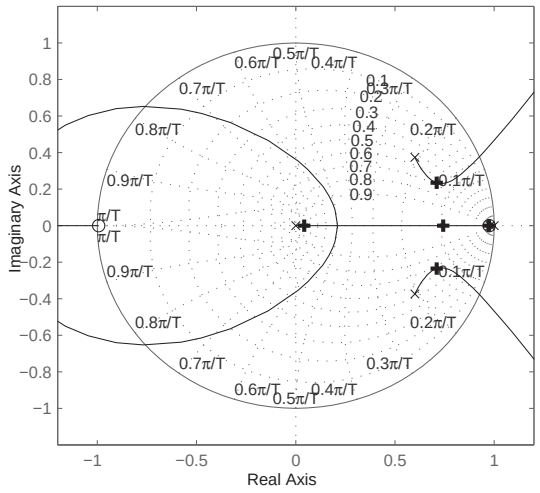


Figure 4.4: Root-locus of the system controlled by the observer-based controller.

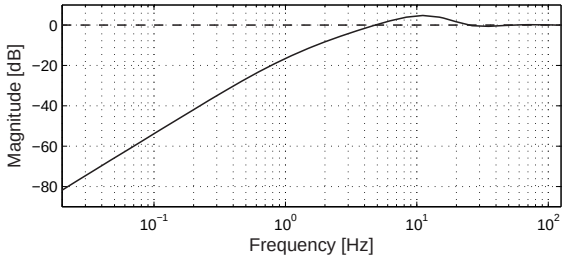


Figure 4.5: Sensitivity of the system controlled by the observer-based controller.

disturbances are sufficiently rejected at least up to 4.0 Hz. Figure 4.4 shows the root-locus of the system with the closed-loop poles indicated with '+' signs for feedback gain 1. To obtain this root-locus we assumed no quantization and estimation errors caused by the Hall sensors in combination with the interpolation mechanism. A plot of the sensitivity, that indicates how well the disturbance d is rejected, is depicted in figure 4.5. In this figure it can be seen that disturbances are rejected up to 4.8 Hz which satisfies the requirement. The feed-forward gain K_{ff} was set to 0.029.

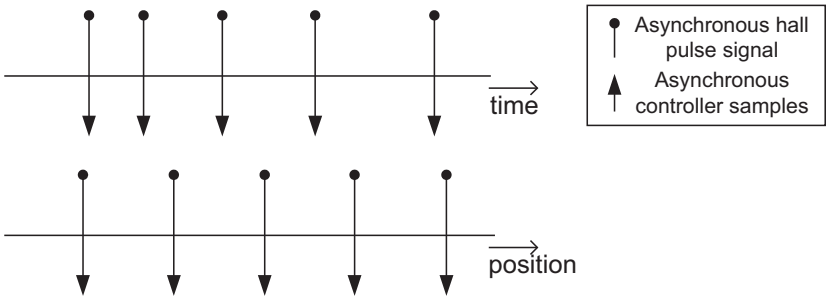


Figure 4.6: Typical pulse sequence for event-driven controller.

The parameters for the $\alpha\beta - tracker$ were mainly chosen on the basis of simulations. When decreasing α and β, the measurement noise from the measured position and estimated velocity is filtered better, i.e. less "confidence" is expressed in the innovation terms. A disadvantage of lowering the values for α and β is the increased delay in the system. Increasing the delay influences tracking performance but also stability in a negative way.

4.4 Event-driven control

For the event-driven controller we propose to execute both the measurement and the control update at the moment of a Hall pulse, as depicted in figure 4.6. When the motor is not running at a constant velocity, the updates are not equidistant in time. It is therefore not possible to use the classical design methods which typically assume that control updates are equally spaced in time.

However, we can apply variations on classical design methods, if we define our models of the plant and the controller in the spatial (angular position) domain instead of the time domain, as initially proposed in [36]. This idea is based on the observation that the Hall pulses arrive equally spaced in the spatial domain, as the Hall sensors have an equidistant distribution along the axis of the motor (which is true up to ± 0.2 rad as explained in section 4.2). To use this reasoning, we first have to transform the motor model as given in equation (4.2) to an equivalent model in which the motor angular position is the independent variable. After that we will show how the controller design can be performed using classical control theory.

4.4.1 Transformation to spatial domain

The transformation ideas are explained in [36]. We will recapitulate the main steps of the transformation for this specific example.

The transformation is performed via the following relation:

$$\frac{d\theta}{dt}(t) = \omega(t) \Rightarrow \frac{dt}{d\theta}(\theta) = \frac{1}{\omega(\theta)}, \tag{4.8}$$

where $\omega(\theta)$ denotes the angular velocity of the motor and $t(\theta)$ denotes the time, respectively, at which the motor reaches position θ. Under the assumption that $\omega(t) \neq 0$ for all $t > 0$, a one-to-one correspondence between θ and t exists and an interchange of their roles is possible. Note that $\omega(t) \neq 0$ is valid under normal operating conditions for the considered example, as the motor does not change direction.

Using (4.8) on (4.2) we obtain the motor model in the spatial domain:

$$\begin{aligned} &\tfrac{dt}{d\theta}(\theta) = \tfrac{1}{\omega(\theta)} \\ &\tfrac{d\omega}{d\theta}(\theta) = \tfrac{1}{J}[-\tfrac{d(\theta)}{\omega(\theta)} - (\tfrac{k^2}{R} + B) + \tfrac{k}{R} \cdot \tfrac{u(\theta)}{\omega(\theta)}] \\ &y(\theta) = t(\theta) \end{aligned} \tag{4.9}$$

where $d(\theta)$ and $u(\theta)$ denote the disturbance torque and the motor voltage, respectively, at motor position θ. Note that time t is now a function of θ and became a state variable in this new description. To consider the disturbance d as a function of the angular position θ is an advantage for many controller designs, as disturbance are often coupled to the angular position, instead of time. This will be further elaborated in section 4.9.2. Moreover, the output $y(\theta)$ is now the time $t(\theta)$ at which the motor reaches position θ.

The error that is input for the feedback controller is now selected to be the difference between the measured time of a Hall pulse ($t_m(\theta)$) and the time at which the Hall pulse ideally should have occurred based on the reference trajectory (which is denoted by $t_r(\theta)$):

$$e_t(\theta) = t_r(\theta) - t_m(\theta). \tag{4.10}$$

This is illustrated in figure 4.7. In this figure it is also shown how the *time error* can be translated into a *position error*. When ω_r is constant and non-zero in the time interval $(t_r(\theta_p), t_m(\theta_p))$, or $(t_m(\theta_p), t_r(\theta_p))$ when $t_m(\theta_p) < t_r(\theta_p)$, where θ_p is the angular position at an encoder pulse detection, then it holds that

$$e_\theta(t_p) = -\omega_r e_t(\theta_p). \tag{4.11}$$

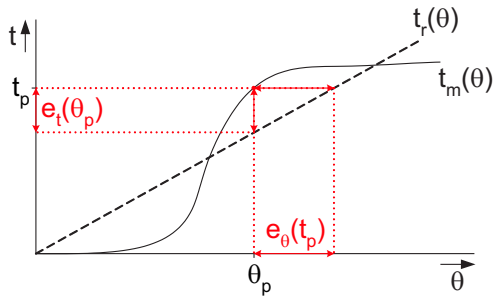


Figure 4.7: Errors in t and θ.

When ω_r is not constant, equation (4.11) can be used as an approximation.

The control objective, that the controller bandwidth should be at least 4.0 Hz, should be translated into a similar requirement in the spatial domain. To define the frequency content of the disturbance, independently of the motor velocity, we analyze the *spatial frequency* $[rad^{-1}]$. The spatial frequency is a characteristic of any structure that is periodic across position in space. It is a measure of how often the structure repeats per unit of distance (completely analogous to "ordinary" frequency with respect to time). The concept of spatial frequency is especially used in wave mechanics and image processing [32]. For the considered disturbance signal, the main component is located at a spatial frequency of $\frac{1}{20 \cdot 2\pi} = 8.0 \cdot 10^{-3}\ rad^{-1}$ (section 4.2). Some discussion on the advantages and disadvantages of control requirements in the spatial domain can be found in section 4.9.2.

4.4.2 Controller design

As can be seen from equation (4.9), the resulting model is non-linear. A common way to deal with such a system is by linearizing the model around steady-state trajectories. The steady-state trajectory is chosen straightforwardly at a constant angular velocity ω_e of the motor after start-up:

$$(t_e, \omega_e, d_e, u_e) = (\frac{1}{\omega_e}\theta, \omega_e, d_e, k\omega_e + \frac{(B\omega_e + d_e)R}{k}) \tag{4.12}$$

where d_e is chosen as the mean value of the disturbance d. The variations around this steady-state trajectory are denoted by $(\Delta t, \Delta\omega, \Delta d, \Delta u)$. Hence, $t = t_e + \Delta t$,

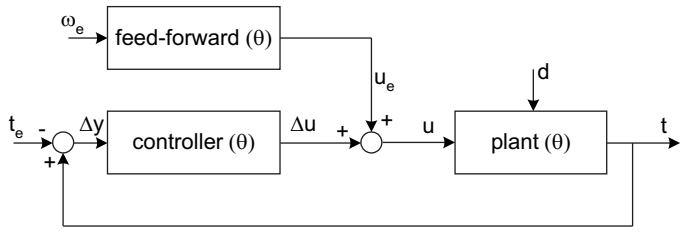


Figure 4.8: Feedback/feed-forward controller structure for linearized dynamics.

$\omega = \omega_e + \Delta\omega$, etc. Around the steady-state trajectory, the linearized dynamics are

$$\begin{aligned} \frac{d\Delta t}{d\theta}(\theta) &= -\frac{1}{\omega_e^2}\Delta\omega(\theta) \\ \frac{d\Delta\omega}{d\theta}(\theta) &= \frac{1}{J\omega_e}[-(\frac{k^2}{R}+B)\Delta\omega(\theta) - \Delta d(\theta) + \frac{k}{R}\cdot\Delta u(\theta)] \\ \Delta y(\theta) &= \Delta t(\theta) \end{aligned} \tag{4.13}$$

We can now design a feedback controller that generates the compensation for the first-order variations, $\Delta u(\theta)$, as defined in (4.13). This is schematically depicted in figure 4.8. The control value applied to the plant should be $u = u_e + \Delta u$. As $u_e = k\omega_e + \frac{B\omega_e R}{k} + \frac{d_e R}{k}$ and we do not know d_e, the feed-forward term takes care of $(k + \frac{BR}{k}) \cdot \omega_e$ and the additional term has to be compensated for by the PD controller. Using the model parameters from section 4.2, we can verify the feed-forward gain that was chosen for the observer-based controller in section 4.3: $K_{ff} = k + \frac{BR}{k} = 0.028 + \frac{3.0\cdot 10^{-5}\cdot 1.0}{0.028} = 0.029$.

We have chosen to design a PD controller (like in the observer-based controller case) and tuned it in the discrete *position* domain. We aim at using a minimal encoder resolution that satisfies the design objectives (section 4.2). The advantage for choosing a minimal encoder resolution is that the software is interrupted as less as possible to perform control updates, which is beneficial for the processor load. We will investigate if we can achieve sufficient control accuracy with an encoder resolution of only 1 PPR. For this we only need to use one of the three Hall sensors available. Choosing this ultimately low resolution has the advantage that we do not have to deal anymore with inaccurate sensor distributions along the motor axis (of ± 0.2 rad). A more extensive discussion about the rationale behind this choice of encoder resolution will be presented in section 4.9.1. To tune the controller, equation (4.13) was first discretized in the spatial domain, using a sample *"distance"* of 2π rad.

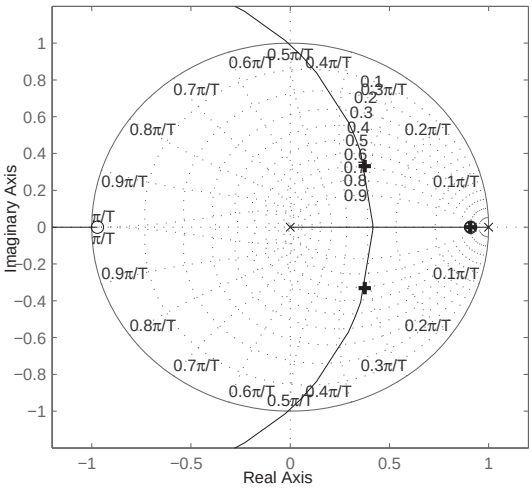


Figure 4.9: Root-locus for $\omega_e = 388$; controller (4.14).

The PD controller was chosen as

$$H_{c1}(\hat{z}) = \omega_t \cdot \frac{(K_p + K_d)\hat{z} - K_d}{\hat{z}} \tag{4.14}$$

where the notation $\hat{z}$ is used instead of z to emphasize that the discretization has been made in the spatial domain, instead of the time domain. Furthermore, this event-driven controller takes the time error as input, while the observer-based controller takes the position error as input. To normalize the controller parameters of both controllers, equation (4.14) is premultiplied by the constant velocity ω_t. The subscript t indicates that the controller is *tuned* for this specific velocity.

To find the values for the controller parameters K_p and K_d, we used the root-locus design method [25]. For $\omega_t = 388$ rad/s and choosing $K_p = 1.0$ and $K_d = 12$ we obtain the root-locus as depicted in figure 4.9. The '+' marks indicate the roots for feedback gain 1.

To evaluate the disturbance rejection for this controller, we computed the sensitivity function as the transfer function from the disturbance $\Delta d(\theta)$ to $\Delta u_d(\theta)$. Both signals are indicated in figure 4.10, which shows a graphical representation of equation (4.13) together with the feedback controller (4.14). Note that $\hat{s}$ is used instead of s to

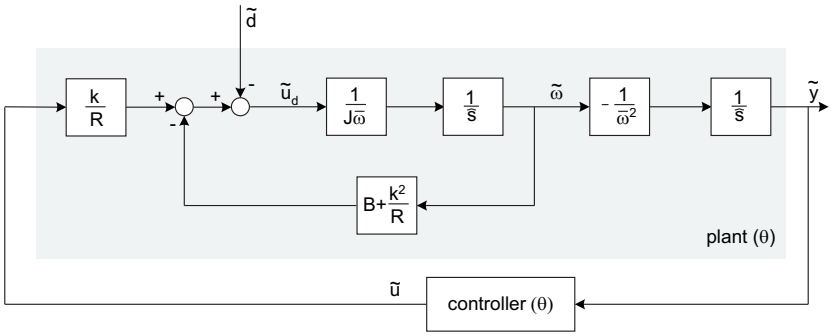


Figure 4.10: Graphical representation of linearized dynamics (4.13).

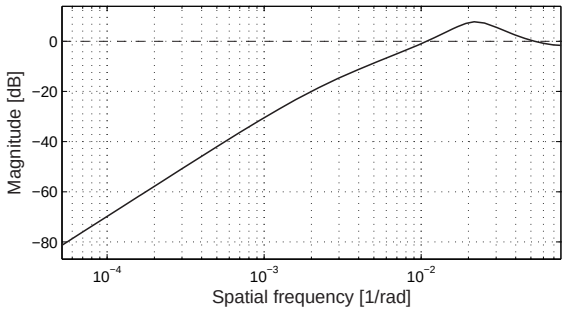


Figure 4.11: Bode magnitude plot of transfer from $\Delta d(\theta)$ to $\Delta d'(\theta)$ for $\omega_e = 388$ rad/s.

emphasize that the integration is performed in the spatial domain. The Bode magnitude plot of this discrete-position transfer is depicted in figure 4.11. We see that spatial frequencies up the $0.01\ rad^{-1}$ are attenuated. This satisfies the required bandwidth as given in section 4.4.1, being $8.0 \cdot 10^{-3}\ rad^{-1}$.

4.4.3 Multiple printing speeds

Now the case is investigated wherein a printer is able to print at multiple printing speeds. For this reason, the system should be analyzed for different values of ω_e. Figure 4.12 depicts the closed-loop poles for $\omega_e = 200, 300, 400$ and 500 rad/s for the above derived controller. It can be seen that for the various values for ω_e the closed-loop system is behaving differently. For $\omega_e = 200$ rad/s the system is even unstable as

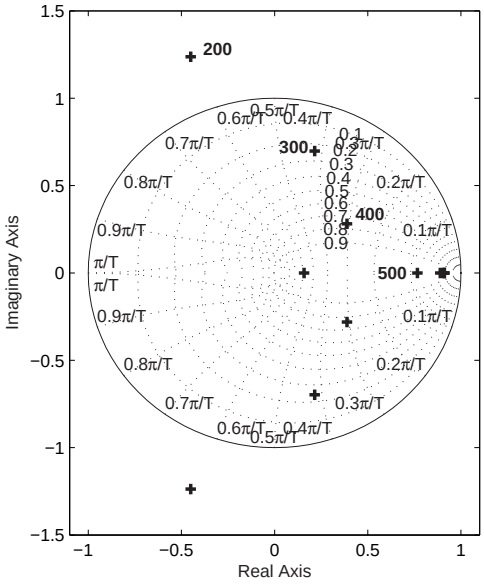


Figure 4.12: Pole locations; $\omega_e = 200, 300, 400$ and 500 rad/s; controller (4.14).

the closed-loop poles lie outside the unit circle.

To solve this problem, we propose to schedule the controller gain proportionally with ω_e, i.e. to use a linear parameter-varying (LPV) controller. When the controller is tuned such that it matches (4.14) for ω_t, we only need to replace ω_t by ω_e in (4.14), while keeping the same values for K_p and K_d:

$$H_{c2}(\hat{z}) = \omega_e \cdot \frac{(K_p + K_d)\hat{z} - K_d}{\hat{z}} \tag{4.15}$$

From the pole locations, as depicted in figure 4.13, we observe that performance is improved, but for $\omega_e = 200$ rad/s the system is close to instability.

A second possibility is to schedule the controller gain in a similar way as proposed in chapter 5, section 5.2. There, a PID controller is designed in continuous-time and then transformed into a discrete-time controller with an approximation method (e.g. Euler, Tustin). With this approximation, the gain for the differential action is divided by the sample time T_s. In the asynchronous application presented in chapter 5 the sample time is kept as a parameter in the PID control algorithm. The "sample

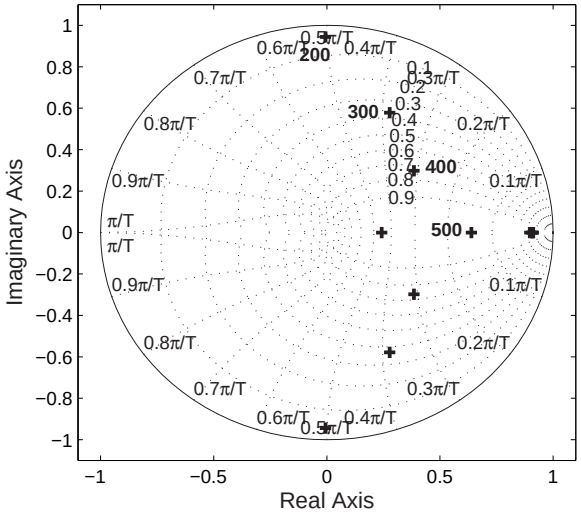


Figure 4.13: Pole locations; $\omega_e = 200, 300, 400$ and 500 rad/s; controller (4.15).

time" T_s is substituted in the algorithm online as the difference between the present and last control update time.

As we have a one-to-one relation between the time and the angular position of the motor axis, we can use the angular velocity and multiply K_d with ω instead of dividing K_d by T_s. In our analysis we multiply K_d with ω_e, as we only analyze the control performance for a specific steady-state velocity. We keep the scheduling as proposed in [36], as the proposed method in section 5.2 also takes the position error as a function of time as input. To again achieve similar performance at ω_t with the same tuning parameters as found before, we compensate K_d by dividing it by the constant ω_t.

$$H_{c3}(\hat{z}) = \omega_e \cdot \frac{(K_p + K_d \frac{\omega_e}{\omega_t})\hat{z} - K_d \frac{\omega_e}{\omega_t}}{\hat{z}} \tag{4.16}$$

The pole locations for the system controlled by controller (4.16) are given in figure 4.14. We observe that for this controller all examined closed-loop poles lie well inside the unit circle. Still, the performance of the controller changes significantly with the choice of ω_e.

To find the controller that creates a system with equal pole locations for any value

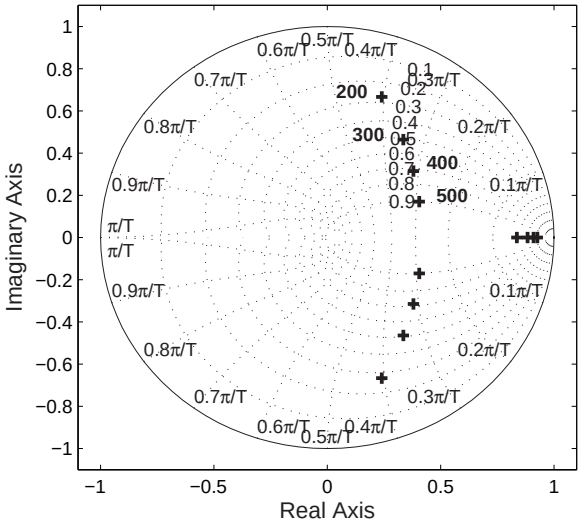


Figure 4.14: Pole locations; $\omega_e = 200, 300, 400$ and 500 rad/s; controller (4.16).

---

of ω_e, we derive the pole locations of the closed-loop discrete position system mathematically, as a function of ω_e. To do so, we first derive for equation (4.13) the transfer function $P(\hat{s})$ from input $\Delta u(\theta)$ to output $\Delta y(\theta)$, which yields

$$P(s) = \frac{\Delta y}{\Delta u} = \frac{1}{\omega_e^2} \cdot \frac{k}{JR\omega_e \hat{s}^2 + (BR + k^2)\hat{s}} \tag{4.17}$$

We then transform (4.17) to the discrete-spatial domain using bilinear approximation. The bilinear approximation, also known as Tustin's approximation [25], maps all the stable poles at the left half of the $\hat{s}$-plane to the interior of the unit circle on the $\hat{z}$-plane with a one-to-one correspondence. Hence, stable $\hat{s}$-plane poles become stable $\hat{z}$-plane poles.

$$P(\hat{z}) = \frac{k}{\pi\omega_e^2} \cdot \frac{\hat{z}^2 + 2\hat{z} + 1}{(JR\omega_e \frac{1}{\pi} + BR + k^2)\hat{z}^2 - 2JR\frac{1}{\pi}\omega_e \hat{z} + JR\frac{1}{\pi}\omega_e - BR + k^2} \tag{4.18}$$

To find the closed-loop poles of the system, we solve $C(\hat{z})P(\hat{z}) = 1$ with the general PD controller $C(\hat{z}) = \frac{(K_p+K_d)\hat{z}-K_d}{\hat{z}}$. After simplification of the equations this gives

us the equality for the pole locations

$$\begin{aligned}&(K_p + K_d)\hat{z}^3 + (2K_p + K_d)\hat{z}^2 + (K_p - K_d)\hat{z} - K_d = \\ &((\alpha\omega_e + \beta)\hat{z}^3 - 2\alpha\hat{z}^2 + (\alpha\omega_e - \beta)\hat{z})\omega_e^2,\end{aligned} \tag{4.19}$$

where $\alpha = JR\frac{1}{\pi^2 k}$ and $\beta = \frac{BR+k^2}{\pi k}$. It can be derived from equation (4.19) that it is not possible to schedule K_p and K_d such that the pole locations are independent of ω_e. We can however "minimize" dependability on ω_e by choosing the controller

$$H_{c4'}(\hat{z}) = \omega_e^2 \cdot \frac{(K_p + K_d\omega_e)\hat{z} - K_d\omega_e}{\hat{z}} \tag{4.20}$$

The closed-loop poles are then found by solving

$$\begin{aligned}&((\alpha - K_d)\omega_e + \beta + K_p)\hat{z}^3 - (2\alpha + 2K_p - K_d\omega_e)\hat{z}^2 + \\ &((\alpha - K_d)\omega_e - \beta + K_p)\hat{z} - K_d\omega_e = 0\end{aligned} \tag{4.21}$$

The controller gains were again chosen such that the poles for $\omega_e = 388$ rad/s match the poles of controller (4.14) for $\omega_e = 388$ rad/s. This results in the controller

$$H_{c4}(\hat{z}) = \frac{\omega_e^2}{\omega_t} \cdot \frac{(K_p + K_d\frac{\omega_e}{\omega_t})\hat{z} - K_d\frac{\omega_e}{\omega_t}}{\hat{z}} \tag{4.22}$$

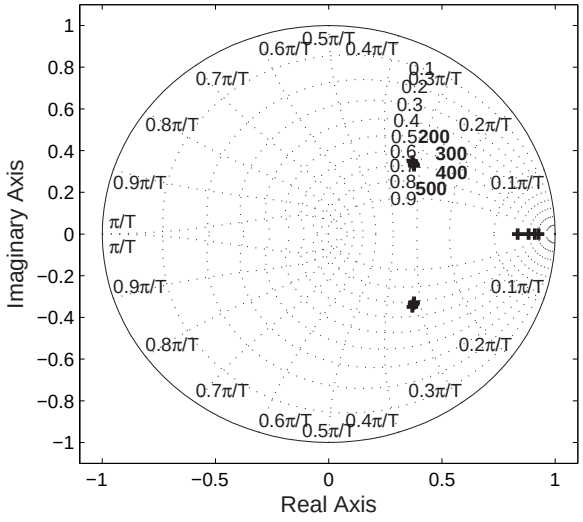


Figure 4.15: Pole locations; $\omega_e = 200, 300, 400$ and 500 rad/s; controller (4.22).

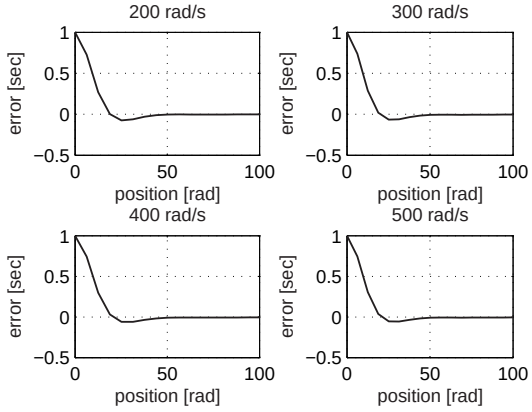


Figure 4.16: Step responses; $\omega_e = 200, 300, 400$ and 500 rad/s; controller (4.22).

Figure 4.15 displays the closed-loop poles for controller (4.22). This controller results in the approximate same control performance for all evaluated values of ω_e *in the spatial domain.* This can be observed in figure 4.16 in which the responses to a unit step are depicted for the four evaluated values of ω_e. The interpretation of a unit step here is that the reference *time* instantaneously changes from 0 to 1 second at position $\theta = 0$ rad. Consequently, the step response displays the time variation (time error, with respect to the steady state trajectory) when the motor reaches a certain position.

As explained earlier (see equation (4.11)), we can transform the error to the time domain from $e_t(\theta)$ by means of equation (4.11), as ω_r is constant after the step has been applied. In the same way we can transform the position scale on the horizontal axis of the step responses to a time scale. The results for the four step responses are plotted in figure 4.17. It can be seen that the settling *time* decreases as ω_e increases. Hence, the developed controller does not have a (constant) settling *time* but at a (constant) settling *distance.* This makes sense for the printer design, as printing accuracy does not vary with the printing speed in this case.

4.5 Implementation issues

When implementing the event-driven controllers proposed above, there are various issues that need to be considered. Some of these issues are caused by the fact that it is not common to implement the event-driven type of controller. The other issues are

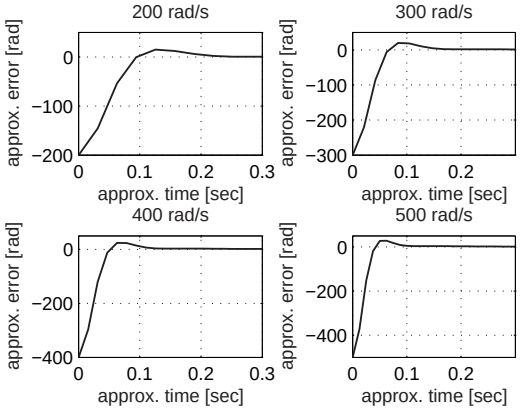


Figure 4.17: Step responses; $\omega_e = 200, 300, 400$ and 500 rad/s; controller (4.22).

common design considerations that also apply for classical control algorithms (when controllers are implemented on processing platforms with finite processing power that is shared with other tasks). Only the most important issues are considered.

4.5.1 Friction

In our system model (4.2) we have not modeled any friction, such as the friction in the bearings of the motor and TTF transportation rollers. Its effect does not have significant influence at higher motor velocities. The effect will mainly influence the start-up and will be considered in simulations. From measurements it turned out that the following relation between the friction torque T_f and motor velocity ω is a good approximation:

$$T_f = 4.6 \cdot 10^{-2} \text{atan}(0.1\omega). \tag{4.23}$$

This is depicted in figure 4.18 for the considered velocities.

4.5.2 Voltage and current limiting

Because of the limited functionality of the amplification circuitry, the current is restricted to $(-15, +15)$ amps. The motor voltage is bounded to $(0, 24)$ volts. From simulations we observe that we only hit the limiting bounds during acceleration and deceleration phases (i.e. when the reference velocity is not constant). As we are mainly interested in disturbance rejection during constant reference velocity, the limiting ef-

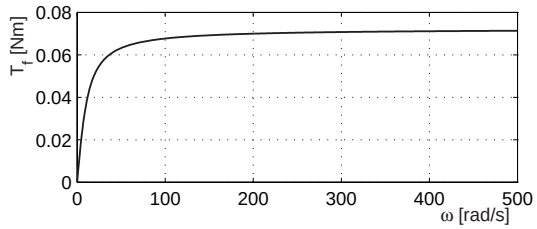


Figure 4.18: Friction torque curve.

fect can be discarded in the analysis.

4.5.3 Time quantization

Because the software on the printer is scheduled by means of a time sliced scheduler (i.e. in a time-driven manner), we can, *unfortunately*, only compute a new control update at a constant frequency, which we refer to as the quantization frequency. As Hall pulses do not arrive at a constant frequency, we have to deal with varying latencies (after all). This phenomenon we call time quantization. Because the quantization frequency is one of the controller design parameters, it is important to consider the influences of this frequency on the control performance. In the next section we will present simulations of the system for various values of this frequency. Section 4.8 gives some discussion on how to select the right value for the quantization frequency. It is important to note that the time-stamping (to obtain ω) is not implemented in software but in digital logic (FPGA), and therefore these time-stamps are not quantized at the same (relatively low) frequency. A much higher frequency (in the order of 10 MHz) is available for this purpose.

4.5.4 Time delay

As computational power on the embedded processor is finite, we will need to take the computation delay between the detection of a Hall pulse and the update of the actuator signal into account. As the controller has to perform the same computation at every update, this computation time can be estimated to be a constant value. From previous implementations we have obtained an estimate of 0.05 msec. We will use this value as a constant time delay in our simulations in the next section.

4.5.5 Start-up

As the event-driven controller is triggered by the Hall pulses, we have a problem when the motor is in stand-still. The controller will not update the motor voltage to a non-zero value as long as no Hall pulses are received. But no Hall pulses will be received if the motor is not rotating. We have solved this by applying a pulse onto the motor voltage to generate the initial start-up, independently of Hall pulses. In the simulations in the next section we have used a start-up pulse with an amplitude of 5 volts and a width of 0.3 seconds. This signal was obtained via trial-and-error, under the conditions that the motor should begin rotating, but when the first couple of pulses are detected, the event-driven controller should take over without being influenced too much by the start-up behavior. Of course, one could choose fancier start-up signals to improve the tracking performance during start-up, but in the considered situation the only controller requirement for the start-up is that the motor is accelerated to a constant velocity in at most 1.3 sec. Note that this start-up pulse also takes care that the motor starts rotating in the right direction, as the direction can not be measured by the 1 PPR encoder.

4.6 Simulation results

Simulations have been carried out for both the observer-based controller as given in (4.3-4.7) and the event-driven controller as given in (4.22), in which a constant reference velocity of 388 rad/s is tracked. At 3 seconds a disturbance torque pulse d, as depicted in figure 4.2, is applied to the system. This pulse resembles the measured disturbance d as induced by a sheet entry at the fuse roll.

The simulation results are depicted in figures 4.19 - 4.22 and show the position error for various simulations with the observer-based controller and the event-driven controller proposed in (4.22) with $e_t(\theta)$ as input.

For ω_e in equation (4.22) we used an estimation of the actual speed, by using the duration from the previous pulse ($t_{p,k-1}$) until the time of the last pulse ($t_{p,k}$), based on the encoder resolution of 2π rad:

$$\omega_e = \frac{2\pi}{\tau_{p,k} - \tau_{p,k-1}} \tag{4.24}$$

From figures 4.19 and 4.20 it can be seen that both controllers perform equally well for the case in which the observer-based controller is running at 250 Hz and 12

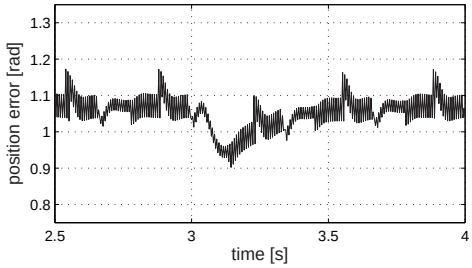


Figure 4.19: Simulation results for the observer-based controller (4.7) with 12 PPR encoder and sample frequency 250 Hz.

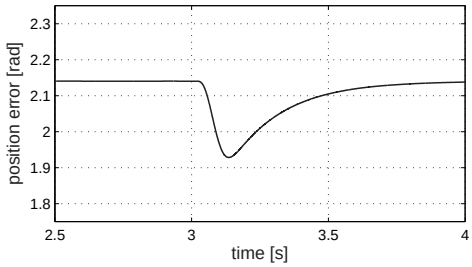


Figure 4.20: Simulation results for the event-driven controller (4.22) with 1 PPR encoder and 500Hz time quantization.

PPR and the event-driven controller running at 1 PPR (resulting in an average sample frequency of 62 Hz) with the quantization frequency set to 500 Hz. Both position errors vary within a range of about ± 0.2 rad. The main difference is that the error is smooth in the event-driven controller situation, but a high frequent ripple is visible in the observer-based situation. This is caused by the small errors that are modeled as inaccuracies in the distribution of Hall pulses over 1 rotation of the motor.

Figure 4.21 shows the position error for the same situation, but here a observer-based controller in combination with a 1 PPR encoder, running at 62 Hz, was used. This frequency was specifically chosen as the event-driven controller also runs at an average frequency of 62 Hz. For this configuration the controller had to be re-tuned, to still obtain a bandwidth of 4 Hz. The following parameter values were obtained: $T_s = \frac{1}{62}s, K_p = 1, K_d = 0.05, \alpha = 1$ and $\beta = 1$. Because we only have 1 PPR, the noisy characteristic due to the inaccurate Hall pulses has disappeared. For this reason both parameters of the $\alpha\beta - tracker$ could be set to one, i.e. no filtering

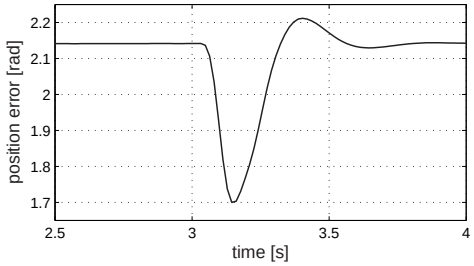


Figure 4.21: Simulation results for the observer-based controller (4.7) with 1 PPR encoder and sample frequency 62 Hz.

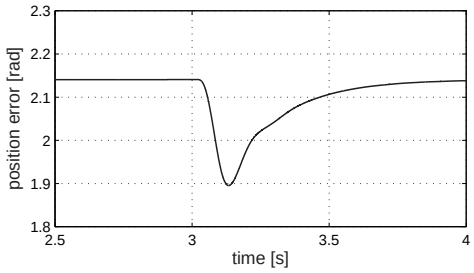


Figure 4.22: Simulation results for the event-driven controller (4.22) with 1 PPR encoder and 50Hz time quantization.

---

was necessary (as can be seen from equation (4.5)). The value for K_d needed to be lowered considerably as otherwise the computed actuator signal would be impossible to realize with the motor amplifier. Comparing figures 4.20 and 4.21 we observe that the event-driven controller outperforms the observer-based controller in this setting, in the sense that the error deviation from the constant equilibrium value is 0.2 rad for the event-driven controller and 0.5 rad for the observer-based controller, which is out of spec.

Figure 4.22 depicts the result of a simulation where we have set the quantization frequency to 50 Hz. Note that this frequency is even lower than the frequency at which Hall pulses arrive. The time delay that is introduced is however never larger than 0.02 seconds. Comparing figures 4.20 and 4.22 it can be seen that the increased delay does not influence the control performance much. Comparing figures 4.21 and 4.22 we see that the event-driven controller at 50 Hz quantization frequency is performing better than the observer-based controller at 62 Hz considering the deviation from the con-

stant error. Apparently, the estimation errors that are made in the linear extrapolation of the observer-based controller during non-constant velocity have stronger negative influence on the controller performance compared to the time delay of the event-driven controller.

4.7 Measurement results

To validate the simulation results of section 4.6, we compared the observer-based controller (4.7) with the event-driven controller as proposed in (4.22) by implementing them both on a complete prototype document printing system (as the one shown in figure 1.3) at Océ technologies BV. Actually, the observer-based controller was already implemented, as this was the original controller used by the printer manufacturer to drive the TTF belt, and therefore acts as a reference controller.

The model used in section 4.2 was already matched with this prototype system. Therefore, the controller parameters obtained from the analysis and synthesis described in section 4.4.2 could be applied *directly* to control the TTF belt in the prototype.

The experimental results for both controllers are given in figures 4.23 and 4.24. These figures show the position error during printing over 5 seconds (after start-up). In this period, 5 sheets are printed at a speed of 80 pages per minute. As the control performance was measured with the position error in rad, the results can be compared with the simulation results (section 4.6). For the event-driven controller implementation a quantization frequency of 500 Hz was chosen. From simulations (figure 4.22) we obtained that comparable control performance could be achieved at a quantization frequency of 50 Hz, which is of course beneficial for the processor load. Unfortunately we were only able to implement the event-driven controller for one quantization frequency as the test time on the prototype was limited.

Comparing the results in figures 4.23 and 4.24 we observe, as expected from the simulations, similar control performance for the observer-based controller and the event-driven controller, both within spec. The maximum deviation from a the average position error is for both controllers about 0.15 rad, which is smaller than 0.25 rad as required (see section 4.2). However, keep in mind that the event-driven controller operates with an encoder with a resolution that is a factor 12 lower than that for the observer-based controller. Furthermore, the observer-based controller runs at a (constant) control sample frequency of 250 Hz and the event-driven controller at a

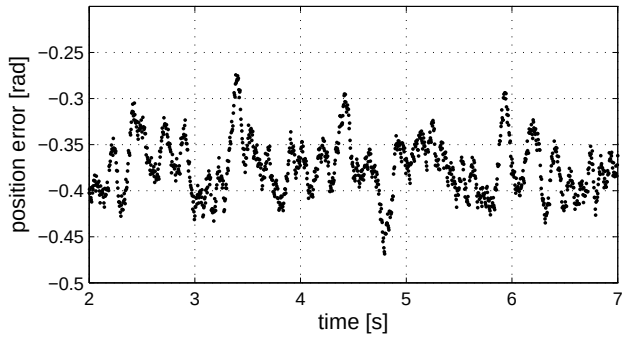


Figure 4.23: Experiment hybrid controller (4.7) with 12 PPR encoder and sample freq. 250 Hz.

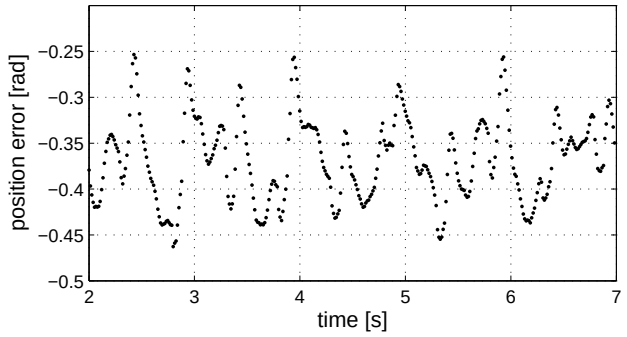


Figure 4.24: Experiment event-driven controller (4.22) with 1 PPR and 500Hz time quantization.

much lower *average* frequency (approximately 62 Hz). The errors caused by the sheet passings can be distinguished in both figures, although there are more disturbances (at different frequencies) acting on the system as can be seen from the measurement data.

4.8 Selection of quantization frequency

From the two event-driven controller simulations in section 4.6 (see figures 4.20 and 4.22) it can be seen that increasing the quantization frequency is beneficial for the control performance. In summary, this is caused by a decrease in the delay between detection of the Hall pulse and update of the actuator signal. Higher values of the

quantization frequency however increase the processor load, which is not desirable. The challenge is to find a specific value for this frequency where a good compromise is achieved between control performance and processor load. Of course, a good compromise for this trade-off has yet to be defined.

Lets take a system level perspective on this trade-off problem. Figure 4.25 depicts the overview diagram of the threads of reasoning technique (see chapter 2), applied to this part of the design. While chapter 2 focusses on the system level design choices, here we zoom in on a specific part of the system design. Nevertheless, we are able to link the reasoning to the main design drivers (section 2.2), as specified for the complete printer, in a concise diagram. For illustration purposes, we kept a solved conflict in the overview diagram. The solution is indicated by means of an extra arrow symbol, as shown in the legend, starting in the conflict arrow.

In the figure it is shown that from the main design driver *cost price* we have obtained that the conventionally applied encoder was too expensive. Therefore, we opted to use the Hall pulses of the brushless DC-motor as position sensor, which makes it impossible to control the motor with the classical control laws, as the sensor resolution is too low to achieve the required control performance. This actually was a conflict in one of the initial iterations through the process of the threads of reasoning technique. We opted two alternative solutions for this conflict: use the observer-based controller or use the event-driven controller. These are indicated as design *choices* in the diagram.

The main design driver *time-to-market* advocates a predictable and composable software behavior. From this perspective, the choice was made for a time sliced scheduler to schedule the tasks on the processing platform (see also section 2.4.2). As all tasks are executed at a constant frequency, it is not straightforward to implement the event-driven controller, which is asynchronous. This gives rise to the first conflict, as indicated in the figure. As a solution to this conflict we already proposed to update the actuator signal at a constant frequency (quantization frequency). As the Hall pulses do not arrive at a constant frequency, we have to deal with varying latencies between receiving Hall pulses and the updates of the control algorithm.

When the quantization frequency is increased, the processor load is increased as well. As we have to implement the controller on an embedded platform with limited processor power, we would like to have the processor load for the control algorithm as low as possible. On the other hand, if we decrease the quantization frequency, we increase the control response time, which has a negative effect on the control perfor-

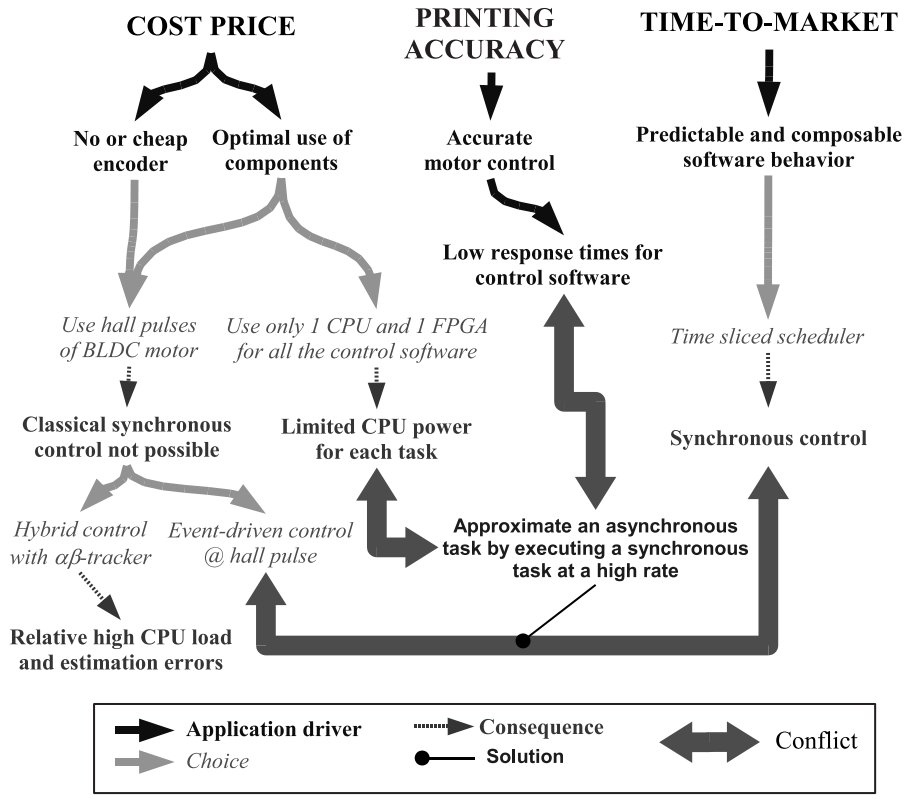


Figure 4.25: Overview diagram of threads of reasoning applied to quantization problem.

mance. This is in conflict with the application driver of accurate motor control to achieve a high printing accuracy. At this point in the overview diagram we identified the trade-off *qualitatively*.

As we are able to choose any value of the quantization frequency (as long as the system requirements are satisfied) we have to choose the value such that a good trade-off is made between the control performance and processor load. Because we have two objectives related to two different disciplines, this is a typical (and relatively easy) example of a multi-disciplinary multi-objective optimization problem. To take a decision for a specific value of the quantization frequency, we need *quantitative* data for both objectives. This is expressed in two simple mono-disciplinary models that express the relation between the quantization frequency and the control performance

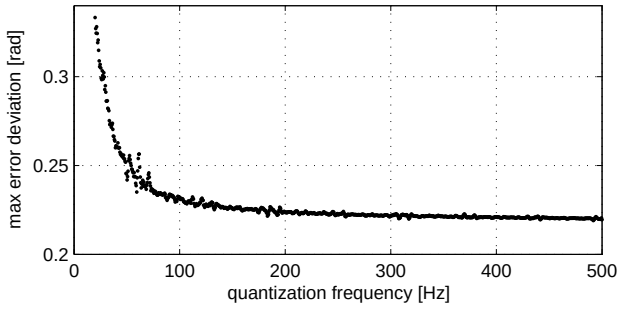


Figure 4.26: Maximum position error deviation as function of the quantization frequency.

and processor load, respectively, as presented in the following two sections. These simple models are based on the complex models including lots of details of the mono-disciplines. Section 4.8.3 gives some discussion on how the trade-off subsequently is made, which partly is still an open issue.

4.8.1 Model of control performance

To obtain an abstract model that describes how the control performance is influenced by the quantization frequency of the event-driven controller implementation, we performed 1000 simulations for values of the quantization frequency in the range $[20, 500]$ Hz. As the measure for control performance, we take the peek-to-peek distance of the position error, for a specific simulation with a specific constant reference velocity to be tracked and a specific disturbance to be rejected. For the simulations we used the exact same situation as described in section 4.6. That is, a constant reference of 388 rad/s to be tracked over 2 seconds, with a disturbance pulse after 1 second, as given in figure 4.2.

The results are depicted in figure 4.26. It can be observed that the maximum error decreases as the quantization frequency increases, as expected. Only below 62 Hz the error deviation starts increasing rapidly, as below 62 Hz not every Hall pulse is followed by a control update anymore. Because the motor is running at 388 rad/s and the resolution of the position sensor is 1 PPR, Hall pulses arrive at $\frac{388}{2\pi} = 62$ Hz. Furthermore, we observe that for these values the results are more noisy, however a clear trend is present.

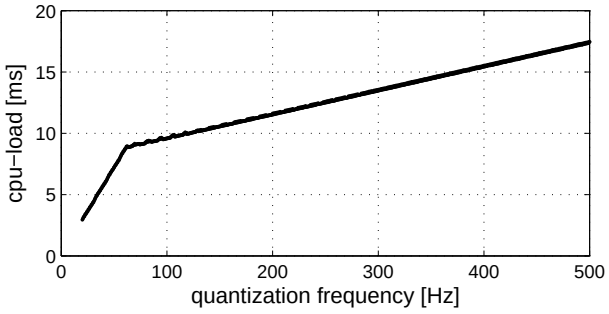


Figure 4.27: Predicted processor load for the event-driven controller as function of the quantization frequency.

4.8.2 Model of processor load

To see the effect of the quantization frequency on the processor load, we predict the time the processor will need to execute the computations for the event-driven control algorithm. For this we need a model of the processor and knowledge on how the algorithm is implemented in software. Chapter 6 describes in detail how to model the processor and how the processor load can be predicted. In this section we will only present the software implementation and the resulting prediction results, without going into full details.

The implementation of the controller is given in the pseudo code below. At a fixed frequency, i.e. the quantization frequency, the code is executed. However, lines 3 and 4 are only executed when a Hall pulse has been detected.

```
1  input(Hall signal);
2  if (Hall signal updated) then
3    update controller;
4    output(actuator signal);
5  end;
```

Lets assume that the software will run on an Infineon XC167 CPU at a clock frequency of 40 MHz. From the processor model we obtain that lines 3 and 4 will take approximately 82 μs and that lines 1, 2 and 5 will take approximately 13 μs.

The processor load prediction results are given in figure 4.27. For this we used the same simulations as described in the previous section, and recorded the number

of checks (lines 1, 2 and 5) as well as the number of control updates (lines 3 and 4). The number of checks can actually be straightforwardly calculated as the quantization frequency times the simulation duration (i.e. 1.5 seconds). To obtain the processor load, these numbers were multiplied by the predicted computation times for each check and each control update.

In figure 4.27 we see that for quantization frequencies above 62 Hz the processor load is linearly increasing. For values below 62 Hz, the increase is also linear but with a larger slope. This is because Hall pulses are arriving at approximately 62 Hz when the motor is running at constant velocity of 388 rad/s. For quantization frequencies below 62 Hz we therefore execute less checks than Hall pulses are received. The values depicted in figure 4.27 can be approximated analytically. For quantization frequencies well below 62 Hz, the processor load is equal to

$$(13 + 85) \cdot 10^{-6} \cdot \text{quantization frequency} \cdot 1.5 \tag{4.25}$$

seconds. For quantization frequencies well above 62 Hz the processor load is equal to

$$13 \cdot 10^{-6} \cdot \text{quantization frequency} \cdot 1.5 + 82 \cdot 10^{-6} \cdot \#\text{ of encoder pulses} \tag{4.26}$$

seconds. In this last expression the number of encoder pulses is equal to $62 \cdot 1.5 = 93$.

4.8.3 Making the trade-off

Figure 4.28 shows both objectives in one graph for various quantization frequencies. This figure clearly depicts the trade-off that has to be made, as only *one point* in this graph will be selected for final implementation. An optimization perspective will be taken on this selection problem.

The probably most well-known multi-objective optimization method is Pareto optimization. This optimization method finds all the solutions for which the corresponding objectives cannot be improved without degradation in another. These solutions are called Pareto-optimal [83, 87]. However, a Pareto optimization yields no single solution. In the presented situation, Pareto-optimal solutions are even found in the complete range of the quantization frequency. This is shown in figure 4.29 that depicts all Pareto-optimal solutions.

To actually select one solution, additional information is required. This additional information is typically represented as weighting values assigned to each of the objectives, thereby reflecting their relative importance in the selection process. After

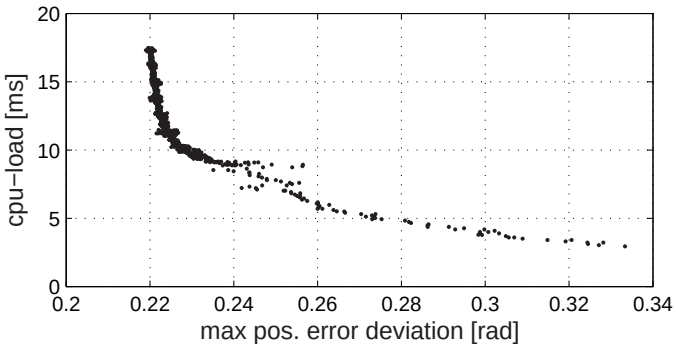


Figure 4.28: Control performance versus processor load as function of the quantization frequency.

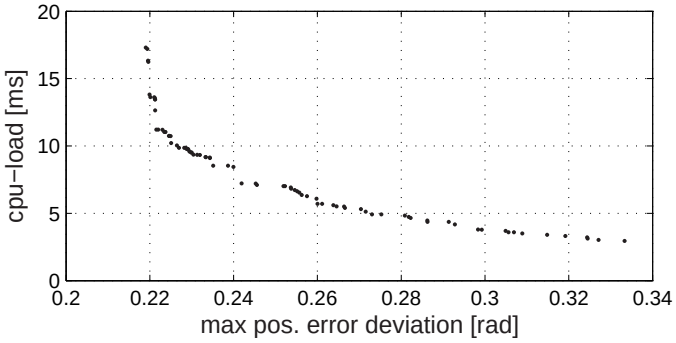


Figure 4.29: Pareto-optimal front.

weighting the objectives, the one solution with minimal "cost" is selected, unless this solution does not satisfy the design requirements. Now we can talk really about an optimal solution.

An open point is still how to choose appropriate weights, as criteria often have different dimensions - "how to compare apples with pears". In most design processes, the system architect is entrusted with this task. Although he has the ability to reason over the multiple disciplines, it is a difficult job that is dominated by subjective arguments. Approaches are needed to make the trade-off decisions in a structured manner. An example of such an approach that relies on both qualitative and quantitative data is

Analytic Hierarchy Process (AHP) [68]. The AHP uses the human ability to compare single properties of alternatives. It helps decision makers in choosing the best alternative, and provides a clear rationale for the choice. It is however still the task of the system architect to feed the process with data that relies on views over the multiple disciplines that are involved in the decision process. Creating structured approaches to compute the "best" solution, based on the quantitative data from the individual disciplines is an ongoing challenge of both industry and academia.

Considering the specific presented problem, one might select the solution around a quantization frequency of 80 Hz, as for higher frequencies the control performance is not improved significantly. However, when processor load is an important objective in the design, and the printing accuracy is already high enough for error deviations below 0.27 rad, one might consider choosing a lower quantization frequency of for instance 40 Hz. When more objectives would play an important role in the trade-off making process, the problem gets more complicated to be solved by such straightforward reasoning and using weighting factors might be more supportive, although selecting them is not straightforward.

4.9 Discussion

4.9.1 Sensor resolution

For the event-driven controller we choose to use only one of the three Hall sensors to obtain a sensor signal with a resolution of 1 PPR. From a physical point of view, this is the lowest encoder resolution one can choose. In section 4.2 we explained that the highest frequency we need to compensate for is located at a spatial frequency of $0.008\ rad^{-1}$ and is caused by the roll that is coupled to the motor and drives the TTF belt. (see section 4.2). According to the Nyquist sampling theorem, we need at least a *spatial sample frequency* of $2 \cdot 0.008 = 0.016\ rad^{-1}$ to measure the signal. However, as we know from the waterbed effect (Bode's sensitivity integral [14]), we need to update the controller at a higher frequency to compensate for the disturbance. This can also be observed in figure 4.11, where the Nyquist sample frequency is located at the right most frequency in the graph. When we tune the controller to attenuate frequencies up to a higher frequency, disturbances at frequencies closer to the Nyquist sample frequency are amplified even more. The 1 PPR encoder gives us a spatial sample frequency of $0.16\ rad^{-1}$, which is 10 times higher than the Nyquist sample frequency, and therefore gave us enough freedom to tune the controller to get satisfactory control

performance.

The rationale behind the low sample frequency was system cost price. From this perspective, the choice was made to implement all control software of the printer paper path on one CPU. To accomplish this, the processor load of every individual task was minimized. By decreasing the sensor resolution, the processor load for the control algorithm was decreased considerably, as the sample frequency of the event-driven controller varies relatively with the sensor data frequency (see section 4.9.3 for discussion on the processor load). Furthermore, when only one of the three sensor signals has to be acquired, we only need 1 input connection in stead of 3. This again potentially reduces system cost price.

When using multiple pulses per revolution it is important to know how the pulses are distributed along one full revolution. Typically, when using the Hall pulses of the brushless DC-motor, we have to deal with inaccuracies in the position of the pulse of ± 0.2 rad (3%). To correct for these errors, we could use some calibration algorithm (see [84]), but this would increase software complexity (and therefore also the processor load) and could possibly also result in additional hardware (if an indexing mechanism would be required). When using only one pulse per revolution we benefit from perfect pulse "distribution", i.e. *exactly* 2π rad in between each two sequential pulses. We, therefore, have *no measurement error*.

4.9.2 Disturbances and resonant frequencies

As stated in section 4.2, the purpose of the implemented controllers is twofold: achieving satisfactory tracking and disturbance rejection. For disturbance rejection it is, in this motion control case study, important to be aware of the fact that disturbances can be categorized in two types:

1. Disturbances having a frequency content that varies with the velocity of the controlled actuator,

2. and disturbances having a frequency content that is independent of the actuator velocity.

In the considered example, most of the disturbances are of type 1. Examples can be found in bearings, axes, rolls, traveling sheets of paper, etc, that all rotate with a velocity that is controlled by the motor. When the motor velocity decreases, all frequencies of the disturbances decrease with the same factor. An example of a type

2 disturbance in this case study could be a vibration in the construction that is caused by another actuator.

As we have seen in this chapter, one has a considerable advantage for type 1 disturbances when designing in the spatial domain, which was applied for the event-driven controller. The reason is that the type 1 disturbances all have a *spatial* frequency content that does not change with the actuator velocity. When type 1 disturbances are to be compensated for with conventional (time-driven oriented) control design, one has to choose a specific steady state (constant velocity) and design the controller for this particular situation. Further analysis should verify whether the required control performance is achieved for all operational velocities. If, on the other hand, we are dealing with a type 2 disturbance in the controller design using the spatial domain, we also have to choose a specific steady state, as the spatial frequency of these disturbances does vary with the motor velocity.

To achieve satisfactory tracking it is important to consider the resonant frequencies of the system. At and around these frequencies the system will not achieve good tracking or even show unstable behavior. Common practice is to design the system such that these frequencies are outside the operational mode, such that the effects can be neglected in analysis and synthesis. However, in some designs the controller has to deal with this limitation of the system. This is often achieved by designing notch filters tuned at the known resonant frequencies, such that they are not excited. When designing an event-driven controller by means of the spatial domain analysis, we cannot simply design a notch in the spatial domain, as its cut-off frequency will change with the velocity of the motor. A solution to this problem is to implement a time-varying notch filter [1].

4.9.3 Processor load

Next to the cost price reduction by using cheap encoders, another important reasons to implement the proposed event-driven controller is to reduce the processor load. In section 4.8.2 it was already predicted what processor power is needed for the event-driven controller implementation. As a measure for the processor load we used the time the processor would need to execute the controller computations for a particular simulation. For this, the simulation data from section 4.6 was used. From figure 4.27 it can be seen that at a quantization frequency of 500 Hz, for which the control performance is approximately equal to the control performance of the observer-based controller, the processor load is 18 ms. For a quantization frequency of 40 Hz, the

processor load is even reduced to less than 6 ms.

The processor load of the event-driven controller furthermore relates to the chosen encoder resolution. When the resolution is increased, the control algorithm would be computed more often. For instance, when a resolution of 2 PPR in combination with a quantization frequency of 500 Hz, would be considered, the processor load would be increased to 26 ms for the considered situation. Fortunately, 1 PPR resolution was sufficient to control the motor according the required control performance. The processor load varies also with the velocity of the motor. When the event-driven controller in combination with the 1 PPR encoder and a quantization frequency of 40 Hz would control the motor at a velocity of 200 rad/s (instead of 388 rad/s as considered above), the processor load would only be 2.3 ms.

For the observer-based controller, we have measured that each control action takes 104 μs. This is slightly more than the computation time of the event-driven controller (which was 13 μs for the check at the quantization frequency and 82 μs for the update of the controller), because of the complexity of the observer-based control algorithm. For the simulation depicted in figure 4.19, in which the observer-based controller was running at 250 Hz, we compute a processor load of $104 \cdot 10^{-6} \cdot 250 \cdot 1.5 = 39$ ms. Therefore, we conclude that the event-driven controller implementation reduces the processor load for the controller task with a factor 2 when the controller is running at a quantization frequency of 500 Hz. This processor load reduction is even increased up to a factor of 6 when the quantization frequency is set to 40 Hz, without deteriorating the control performance too much.

4.10 Conclusions

In this chapter we presented the use of event-driven control to accurately control a brushless DC-motor on the basis of a Hall encoder having a resolution of 1 pulse per revolution. We considered an industrial control problem with a strong eye towards cost price reduction. By means of analysis, simulation and experiments we showed that the performance of the controller satisfied the requirements. Furthermore, we showed that similar control performance was achieved compared to the industrial observer-based controller. However, the event-driven controller was running at a much lower average sample frequency, and therefore involving a significant lower processor load. The advantages of the event-driven controller over the observer-based controller can be summarized as follows:

- Only a cheap encoder with a resolution of 1 PPR necessary, instead of 12 PPR
- Low computational load for the processor
- Enables reasoning in the spatial domain
- Lower number of tuning parameters

Hence, the system performance, as defined in the research hypothesis (chapter 1), was improved considerably with this event-driven controller implementation.

To better suit the time sliced scheduler, we mapped the controller to an implementation in which the actuator signal was updated at a constant sample period. Compared to the originally implemented observer-based controller implementation, a processor load reduction could be obtained up to a factor 6 with the event-driven controller.

The analysis and controller synthesis were based on the observation that the controller triggering is synchronous in the spatial domain. By transforming the system equations from time domain to the spatial domain, we were able to write the control problem, that was asynchronous in the time domain, as a synchronous problem in the spatial domain. With this, we were able to apply classical control theory to design and tune the controller. The obtained control parameters could directly be applied to the implementation. Experiments on a prototype printer validate the results obtained from analysis and simulations.

The resulting control performance, as obtained from the analysis, is defined in the spatial domain. When disturbances are also acting in the spatial domain, which is often the case, we can easily determine how these disturbances are rejected. Furthermore, we can now use the settling *distance* as a control performance measure, instead of the settling *time*. In many cases, also tracking requirements are better defined in the spatial domain. It is however still possible to use approximated values in the time domain by using a constant (or maximum) reference velocity.

As a systems engineering contribution, this chapter also presents the abstraction from the mono-disciplinary details to simple models (expressed in the figures 4.26 and 4.27) that can be used by the system architect to make system level design choices. This is an important step in the multi-disciplinary design of a system as all detailed design choices at the low level contribute to the performance of the complete system. It is still an open issue to create a structured approach to compute the “best” solution for each specific situation, based on the quantitative data from the individual disciplines, and is an ongoing challenge of both industry and academia.

5

Event-driven control to reduce resource utilization[1]

5.1 Introduction

Most research in control theory and engineering considers periodic or time-triggered control systems where continuous time signals are represented by their sampled values at a fixed sample rate. Although it seems in many situations logical to study and implement event-driven controllers, their application is scarce in both industry and academia. A major reason why time-driven control dominates is the difficulty involved in developing a system theory for event-driven systems. Typically, the control community tries to circumvent the event-driven nature of many control systems in order to still use the system theory for time-driven systems. In case the sensors are event-based (i.e. the measurement data arrives not equidistantly in time) often one designs asynchronous observers that provide estimates of the state variable of the plant at equidistant times. For instance, in [16, 31, 50] approaches based on Kalman filter are used, while in [60] a Luenberger-type observer is applied. Since estimates of the state are available at a constant sample rate, standard (state feedback) control analysis and design methods can be applied. Another solution by the control community is to simply assume (or impose as stringent conditions for the software engineers) that

[1] This chapter is partially based on the work that appeared in the proceedings of the American Control Conferences 2005 [70] and 2006 [38], and is submitted for journal publication [40].

the real-time platforms used for implementing controllers are able to guarantee deterministic sample intervals. In reality this is, however, seldom achieved. Computation and/or communication delays of networked control systems [10, 46, 61, 86] are inevitable and hard to predict due to the presence of caches and instruction pipelines in modern hardware platforms and the interference between tasks on processors. The delay and jitter introduced by the computer system can degrade the performance significantly (see for instance [10]). To study these effects in control loops, in [17] the tools Jitterbug and Truetime are advocated, which are mainly based on stochastic analysis and simulations. Other approaches adapt the periodic sampled data theory to incorporate the presence of latencies (delays) or jitter (variations on delays) in servo-loops in the control design. Typically, in this line of work (see e.g. [6, 18, 45, 54, 86]) the time variations in the "event-triggering" can be considered as disturbances and one designs compensators that are robust to it. In [76, 77] time-varying sample times are considered. However, only a *finite* number of possible sample times are allowed. Then one designs controllers and observers that use feedback or observer gains that depend on the *known* sample time. Stability of the closed-loop is guaranteed via the existence of a common quadratic Lyapunov function. However, knowing the (future) sample time is unrealistic in various cases including event-driven control. Moreover, in the event-driven context as proposed here a common Lyapunov function does not exist as asymptotic stability cannot be achieved. Ultimate boundedness [12, 13] with small bounds (a kind of practical stability) is the most one can achieve.

There is another fundamental difference between the previously mentioned work and the current chapter. We will study event-driven controllers for which we design *both* the control algorithm and the way the events are generated that determine when the control values are updated. This is in contrast with the approaches in [6, 18, 45, 54, 76, 77, 86], where the variations in the sample times are considered as externally imposed disturbances. In this chapter the selection of the event-triggering mechanism suits a clear purpose: lowering the resource utilization of its implementation while maintaining a high control performance. The approach taken here is to update the control value *only* when the (tracking or stabilization) error is larger than a threshold and holding the control value if the error is small. Event-driven control strategies [4, 5, 22] have been proposed before to make such a compromise between processor load and control performance. This is also the case for the controller presented in chapter 4. However, the work presented in this chapter is the first that provides a mathematical analysis of such controllers.

To be more precise, this chapter provides theory and insight to understand and tune a particular type of event-driven controlled linear systems. The performance of these novel control strategies is addressed in terms of ultimate boundedness and guaranteed speed of convergence [12]. Depending on the particular event-triggering mechanism used for the control updates, properties like ultimate boundedness for the *perturbed event-driven linear system* can be derived either from a *perturbed discrete-time linear system* or from a *perturbed discrete-time piecewise linear (PWL) system*. Since results for ultimate boundedness are known for discrete-time linear systems, see e.g. [12, 13, 47, 48, 64], and piecewise linear systems, see e.g. [51, 63], these results can be carried over to event-driven controlled systems. In this way we can tune the parameters of the controller to obtain satisfactory control performance on one hand and low processor/communication load on the other.

The outline of the chapter is as follows. In section 5.2 we present two numerical examples that show the potential of the proposed event-driven controllers for reducing resource utilization while maintaining a high control performance. After introducing some preliminaries in section 5.3, we present the problem formulation in section 5.4. In section 5.5 the main approach is given, while the main results are presented for two particular types of event-triggering mechanisms in section 5.6 (the non-uniform case) and 5.7 (the uniform case). Section 5.8 shows how the intersample behavior can be included in the analysis. In section 5.9 it is indicated how the main results can be exploited to compute ultimate bounds for event-driven linear systems in combination with existing theory for linear and piecewise linear systems. Based on these results, we develop tuning rules for event-driven controllers as explained in section 5.10. In section 5.11 we present some examples that illustrate the theory and we end with the conclusions.

5.2 Motivating examples

To show the potential of event-driven control with respect to reduction of resource utilization we present two examples; one academic example using state feedback and one event-driven PID controller with the aim of velocity tracking for a DC-motor, a situation occurring often in industrial practice.

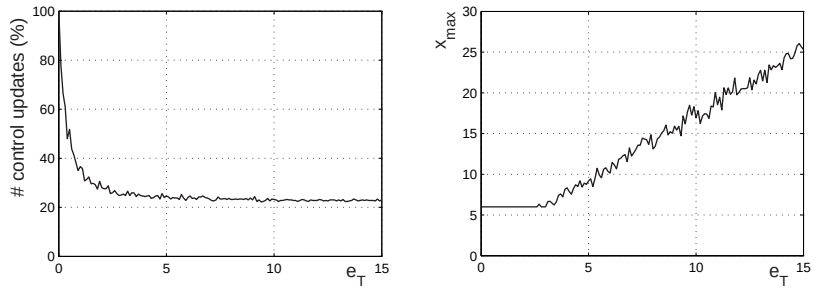


Figure 5.1: e_T versus the control effort and $x_{\max}$ for system (5.1)-(5.2).

5.2.1 Scalar state feedback example

Consider the following simple continuous-time plant

$$\dot{x}(t) = 0.5x(t) + 10u(t) + 3w(t) \tag{5.1}$$

with $x(t) \in \mathbb{R}$, $u(t) \in \mathbb{R}$ and $w(t) \in \mathbb{R}$ the state, control input and disturbance at time $t \in \mathbb{R}_+$, respectively. The additive disturbance satisfies $-10 \leq w(t) \leq 10$. This system will be controlled by a discrete-time controller

$$u_k = \begin{cases} -0.45x_k, & \text{if } |x_k| \geq e_T \\ u_{k-1}, & \text{if } |x_k| < e_T, \end{cases} \tag{5.2}$$

that runs at a fixed sample time of $T_s = 0.1$ time units. Hence, $x_k = x(kT_s)$ for $k = 0, 1, 2, \ldots$. Here, e_T denotes a parameter that determines the region $\mathcal{B} := \{x \in \mathbb{R} \mid |x| < e_T\}$ close to the origin in which the control values are *not* updated, while outside $\mathcal{B}$ the control values are updated in a "normal fashion." In this chapter we will refer to this situation as *uniform* sampling. We will also consider the *(locally) non-uniform* case where reaching the boundary of $\mathcal{B}$ will be the event trigger - in addition to a fixed update rate outside $\mathcal{B}$ - for updating the control values. Figure 5.1 displays the ratio of the number of control updates in comparison to the case where the updates are performed each sample time (i.e. $u_k = -0.45x_k$ for all $k = 0, 1, 2, \ldots$) and the maximal value of the state variable (after transients) $x_{\max} := \limsup_{t\to\infty} |x(t)|$, respectively, versus the parameter e_T. The results are based on simulations. Hence, one can reduce the number of control computations by 75% without degrading the control performance in terms of ultimate bound $x_{\max}$ drastically (e.g. take $e_T = 3$).

5.2.2 Event-driven PID controller

The most common controller in industry is still the PID (Proportional Integral Derivative) controller. We will present an event-driven version of it. Also in [4] an event-driven PID controller is proposed. However, a different event triggering mechanism is used and the presence of noise is disregarded. Moreover, a formal analysis is not presented in [4].

A standard transfer function for a continuous-time PID algorithm is

$$C(s) = K_p + K_i/s + K_d s L(s) \tag{5.3}$$

with $L(s)$, a low-pass filter to deal with high frequency measurement noise. The transfer function of this filter with a bandwidth ω_d is given in (5.4).

$$L\,(s) = \frac{\omega_d}{s + \omega_d} \tag{5.4}$$

To use this controller in a discrete-event environment, the first step is to discretize the transfer function of the controller. This can be done by means of approximation formulas. A common choice for approximating the integral part is to use Forward Euler. To approximate the derivative part in combination with the filter $L(s)$, the Tustin approximation [25] is used. The resulting transfer function of the PID controller in discrete-time is given in (5.5).

$$C\,(z) = K_p + K_i \frac{T_{s,k}}{z-1} + K_d \frac{2\omega_d}{2 + \omega_d T_{s,k}} \frac{z-1}{z + \frac{\omega_d T_{s,k} - 2}{\omega_d T_{s,k} + 2}} \tag{5.5}$$

The controller in (5.5) is suitable as an event-driven controller, with $T_{s,k}$ as a varying sample time and $\tau_k := \sum_{j=0}^{k-1} T_{s,j}$ is the time instant at which the k-th control update is performed. We call τ_k, $k = 0, 1, 2, \ldots$ the *control update times*. Note that in conventional time-driven control $T_{s,k} = T_s$ is fixed and the control update times are equally spaced in time, but for event-driven control it is allowed to change over time. In [4] it is shown that adapting $T_{s,k}$ in (5.5) for every control update improves the *control* performance considerably although it requires additional control computations and thus a bit higher processor load (in comparison to the case where $T_{s,k}$ is kept constant as a kind of "average sampling period").

The event-triggering mechanism (selecting τ_k) is based on the tracking error as follows.

$$\tau_{k+1} = \inf\{t \geq \tau_k + T_{s,\min} \mid |e(t)| \geq e_T\} \tag{5.6}$$

in which $e_T > 0$ is a threshold value and $T_{s,\min} > 0$ is the minimum sample time. We will refer to (5.6) as the *locally non-uniform* mechanism as opposed to the first example. This mechanism uses a uniform sample time of $T_{s,\min}$ for large tracking values $e(t)$, but only when $|e(t)| < e_T$ the control value (so "locally") is held longer and the sample time varies. For shortness we refer to this mechanism as non-uniform in the remainder of the chapter.

The event-driven PID controller is used in simulations to control the angular velocity of a DC-motor. These results will be compared to a standard time-driven PID controller. A simplified motor model is taken with input the motor voltage and output the velocity of the motor axis. The transfer function is given by

$$P(s) = \frac{A}{(\tau_1 s + 1)(\tau_2 s + 1)} \tag{5.7}$$

with the time constants τ_1 and τ_2 equal to 0.33 s and 0.17 s, respectively. The static gain A is 10 rad/Vs.

The gains of the continuous-time controller in (5.3) are determined by loop-shaping (see e.g. [26]) in $K_p = 30\ Vs/rad$, $K_i = 40\ V/rad$ and $K_d = 2\ Vs^2/rad$. In industrial practice, the sample frequency of the time-driven controller is often chosen approximately 20 times the bandwidth of the open-loop system, see e.g. [25]. This bandwidth, defined as the zero-dB crossing of the open loop amplification, is $57\ Hz$ in the considered example. The sample frequency is therefore chosen to be $1\ kHz$. To improve the performance of the controller, a feedforward term was added that feedsforward the set-point speed multiplied by a gain of $\frac{1}{A} = 0.1\ Vs/rad$. Furthermore, the output of the controller is saturated at $+10\ V$ and $-10\ V$. The bandwidth of the low-pass filter f_d is chosen to be $200\ Hz$ (and thus $\omega_d = 2\pi \cdot 200$ rad/s).

Various simulations have been carried out with the reference velocity shown in figure 5.2. In figure 5.3, simulations of the standard time-driven PID controller (5.5) with $T_{s,k} = T_s$ fixed and equal to $1\ ms$ and the event-driven controller (with adaptation of $T_{s,k}$) are shown. For comparison, the parameter e_T of equation (5.6) is chosen such that the maximum error of the event-driven simulation approximates the maximum error obtained from the time-driven simulation. The value of $T_{s,\min}$ is chosen the same as the sample time of the time-driven controller. The values are $e_T = 5 \cdot 10^{-4}$ rad/s and $T_{s,\min} = 0.001$ s. As can be seen from figure 5.3, the event-driven controller does not realize a zero tracking error (in contrast with the time-driven controller). However, in most industrial applications there are often only requirements given for the

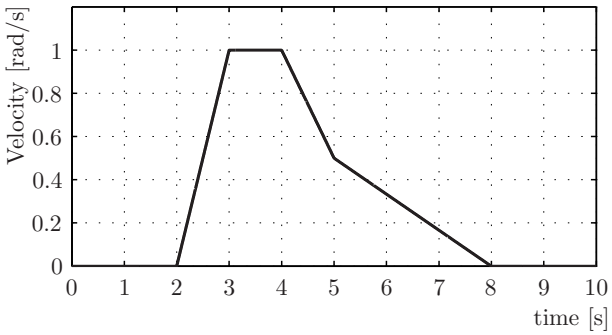


Figure 5.2: Reference signal.

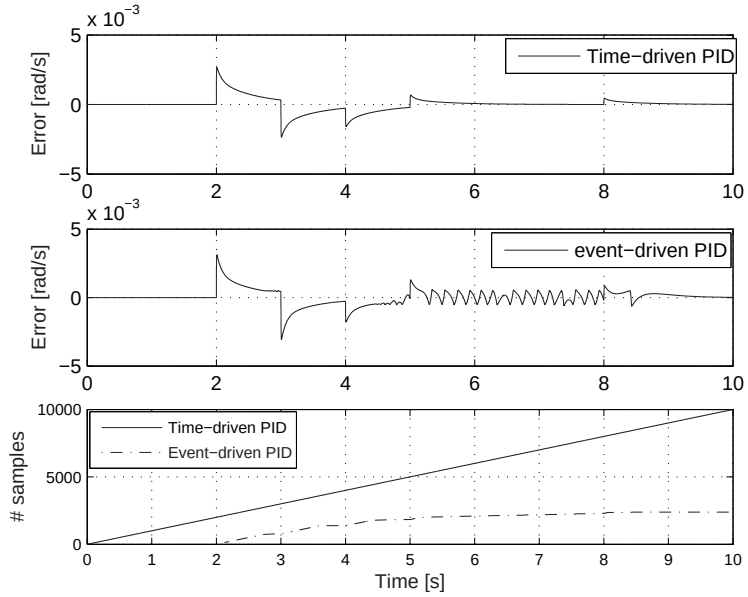


Figure 5.3: Simulation results of time-driven and event-driven simulation.

maximum value of the error.

The third plot in figure 5.3 shows the number of samples that is needed for the control algorithms. This amount is equal to 10,000 for the time-driven controller as it is running on a constant sample frequency of 1 kHz for 10 seconds. The number

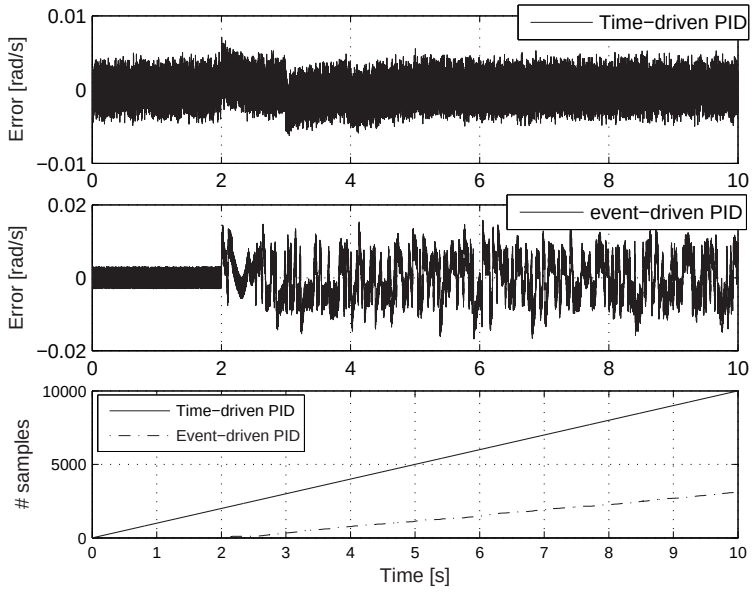


Figure 5.4: Simulation results with measurement noise added.

of samples needed for the event-driven controller based on (5.6) is 2400, leading to a reduction of 76% in the number of control updates.

In the next simulations uniformly distributed measurement noise is added to the output of the process in the range of $[-0.003, 0.003]$ rad/s that is at maximum 3% of the maximal measured velocity. To obtain the best results with the event-driven controller, the value of e_T needs to be increased to make sure it will not be triggered continuously by the noise. The new value is $e_T = 7 \cdot 10^{-3}$ rad/s. The results of the simulations with measurement noise included are depicted in figure 5.4. A slightly worse performance of the event-driven controller compared to the time-driven controller is to be accepted here, especially considering the reduction of control updates to less than 3100 (69% reduction).

Of course, the reduction in control updates has to be related to its effect on resource utilization especially since the event-triggering mechanism creates some overhead as well. Depending on the ratio between the (on-line) computational complexity of the control algorithm, the overhead of the event triggering mechanisms and i/o access of

the processor, the reduction of control computation indeed lowers the processor load considerably.

Both examples indicate the potential benefits of event-driven control in practice. However, as already mentioned in the introduction, a formal analysis of this type of controllers is missing in the literature, which hampers the exploitation of event-driven control. To contribute to fill this gap, we will analyze state feedback controllers using the uniform sampling of section 5.2.1 and the (locally) non-uniform sampling of (5.6). We will focus on the stabilization problem. A precise problem formulation will be given in section 5.4 after introducing some preliminaries next.

5.3 Preliminaries

For a matrix $M \in \mathbb{R}^{n\times m}$, we denote $M^T \in \mathbb{R}^{m\times n}$ as its transposed. A matrix $M \in \mathbb{R}^{n\times n}$ is called positive definite, denoted by $M > 0$, if for all $x \in \mathbb{R}^n$ with $x \neq 0$ it holds that $x^T M x > 0$. For a set $\Omega \subseteq \mathbb{R}^n$ we denote its interior, its closure and its boundary by intΩ, clΩ and $\partial\Omega$, respectively. For two sets Ω_1 and Ω_2 of $\mathbb{R}^n$, the set difference $\Omega_1 \setminus \Omega_2$ is defined as $\{x \in \Omega_1 \mid x \notin \Omega_2\}$ and the Minkowski sum as $\Omega_1 \oplus \Omega_2 := \{u + v \mid u \in \Omega_1, v \in \Omega_2\}$. The complement of $\Omega \subset \mathbb{R}^n$ is defined as $\mathbb{R}^n \setminus \Omega$ and is denoted by Ω^c.

Consider a time-varying discrete-time system

$$x_{k+1} = f(k, x_k, w_k) \tag{5.8}$$

with $x_k \in \mathbb{R}^n$ the state and $w_k \in \mathcal{W}_d$ the disturbance at discrete-time $k \in \mathbb{N} := \{0, 1, 2, \ldots\}$ or a time-varying continuous-time system

$$\dot{x}(t) = f(t, x(t), w(t)) \tag{5.9}$$

with $x(t) \in \mathbb{R}^n$ the state variable and $w(t) \in \mathcal{W}_c$ the disturbance at time $t \in \mathbb{R}_+$. $\mathcal{W}_c$ and $\mathcal{W}_d$ denote the disturbance sets, which are assumed to be convex, compact and contain 0. We define the set $\mathcal{L}_1([t_1, t_2] \mapsto \mathbb{R}^p)$ as the Lebesgue space of integrable functions on $[t_1, t_2]$ to $\mathbb{R}^p$ and $\mathcal{L}_1^{\text{loc}}([t_1, t_2] \mapsto \mathbb{R}^p)$ as the Lebesgue space of locally integrable functions from $[t_1, t_2]$ to $\mathbb{R}^p$. Similarly, $\mathcal{L}_1^{\text{loc}}([t_1, t_2] \mapsto \mathcal{W}_c)$ denotes all $\{w \in \mathcal{L}_1^{\text{loc}}([t_1, t_2] \mapsto \mathbb{R}^p) \mid w(t) \in \mathcal{W}_c \text{ for almost all } t \in [t_1, t_2]\}$. Analogously, for discrete-time signals we write $\mathcal{W}_d^\infty$ for the set of infinite sequences given by $\{(w_0, w_1, w_2, \ldots) \mid w_k \in \mathcal{W}_d,\ k \in \mathbb{N}\}$.

Definition 5.1 (Robust positive invariance) *The set $\Omega \subseteq \mathbb{R}^n$ is a robustly positively invariant (RPI) set for the discrete-time system* (5.8) *with disturbances in $\mathcal{W}_d$, if for any $x \in \Omega$, $k \in \mathbb{N}$ and any $w \in \mathcal{W}_d$ it holds that $f(k, x, w) \in \Omega$. The set $\Omega \subseteq \mathbb{R}^n$ is a robustly positively invariant (RPI) set for the continuous-time system* (5.9) *with disturbances in $\mathcal{W}_c$, if for any time $t_{ini} \in \mathbb{R}_+$, any state $x_{ini} \in \Omega$ and any disturbance signal $w \in \mathcal{L}_1^{\text{loc}}([t_{ini}, \infty) \mapsto \mathcal{W}_c)$ it holds that the corresponding system trajectory satisfies $x(t) \in \Omega$ for all $t \geq t_{ini}$.*

Definition 5.2 (Ultimate boundedness) [12] *We define that the discrete-time difference equation* (5.8) *is* ultimately bounded *(UB) to the set Ω, if for each $x_0 \in \mathbb{R}^n$ there exists a $K(x_0) > 0$ such that any state trajectory of* (5.8) *with initial condition x_0 (and any arbitrary realization of the disturbance $w : \mathbb{N} \mapsto \mathcal{W}_d$) satisfies $x_k \in \Omega$ for all $k \geq K(x_0)$. Similarly, we call* (5.9) ultimately bounded *(UB) to the set Ω, if for every initial condition $x_0 \in \mathbb{R}^n$ there exists a $T(x_0) > 0$ such that any state trajectory of* (5.9) *with initial condition $x(0) = x_0$ (and any arbitrary realization of the disturbance $w :\in \mathcal{L}_1^{\text{loc}}([0, \infty) \mapsto \mathcal{W}_c)$) satisfies $x(t) \in \Omega$ for all $t \geq T(x_0)$.*

5.4 Problem formulation

We consider the system described by

$$\dot{x}(t) = A_c x(t) + B_c u(t) + E_c w(t), \tag{5.10}$$

where $x(t) \in \mathbb{R}^n$ is the state, $u(t) \in \mathbb{R}^m$ the control input and $w(t) \in \mathcal{W}_c$ the unknown disturbance, respectively, at time $t \in \mathbb{R}_+$. $\mathcal{W}_c \subset \mathbb{R}^p$ is a convex and compact set, which contains the origin. $A_c \in \mathbb{R}^{n \times n}$, $B_c \in \mathbb{R}^{n \times m}$ and $E_c \in \mathbb{R}^{n \times p}$ are constant matrices. The control goal, which will be made more precise soon, is a "practical stabilization problem" in the sense of controlling the state to a region close to the origin and keep it there irrespective of the presence of disturbances. Note that asymptotic stability cannot be obtained due to the possible persistence of the disturbances.

As a controller for system (5.10) a discrete-time state-feedback controller with gain $F \in \mathbb{R}^{m \times n}$ is considered, i.e.

$$u_k = F x_k, \tag{5.11}$$

where $x_k = x(\tau_k)$, $u_k = u(\tau_k)$ using the zero-order hold $u(t) = u_k$ for all $t \in$

$[\tau_k, \tau_{k+1})$. Hence, the system is given by

$$\dot{x}(t) = A_c x(t) + B_c u(t) + E_c w(t) \tag{5.12a}$$

$$u(t) = Fx(\tau_k), \qquad \text{for } t \in [\tau_k, \tau_{k+1}) \tag{5.12b}$$

The *control update times* τ_k are in conventional time-driven control related through $\tau_{k+1} = \tau_k + T_s$, where T_s is a fixed sample time meaning that the control value is updated every T_s time units according to (5.11). To reduce the number of required control calculations, we propose not to update the control value if the state $x(\tau_k)$ is contained in a set $\mathcal{B}$ close to the origin. As such, we consider a set $\mathcal{B}$ that is open[2], bounded and contains the origin. We will consider two event-triggering mechanisms, which were already discussed in section 5.2.

- The *non-uniform* mechanism as used in the example of section 5.2.2 is given by

$$\tau_1 = \inf\{t \geq \tau_0 \mid x(t) \notin \mathcal{B}\} \text{ and } \tau_{k+1} = \inf\{t \geq \tau_k + T_s \mid x(t) \notin \mathcal{B}\},\ k > 0. \tag{5.13}$$

- The *uniform* mechanism as used in the example of section 5.2.1 is given by

$$\tau_{k+1} = \inf\{jT_s > \tau_k \mid j \in \mathbb{N},\ x(jT_s) \notin \mathcal{B}\}. \tag{5.14}$$

We will use $\tau_0 = 0$ as the first control update time (irrespective if $x(0) \in \mathcal{B}$ or not). In the non-uniform case we have a kind of "start-up behavior" as τ_1 is defined a bit differently than τ_{k+1} for $k > 0$. In both the uniform and the non-uniform case the system is controlled with a fixed sample time T_s when the state $x(t)$ is far away from $\mathcal{B}$. In the uniform case still every T_s time units it is checked, whether or not the state $x(jT_s)$ lies in $\mathcal{B}$ and the set of control update times is a subset of $\{jT_s \mid j \in \mathbb{N}\}$. The latter set can be considered the collection of *control check times*. The non-uniform case does not have this constant checking rate, but has locally (inside $\mathcal{B}$) a non-uniform character as new control updates are triggered by reaching the boundary of $\mathcal{B}$. It might be the case that for certain initial conditions $x(0) = x_0$ and disturbance signals $w \in \mathcal{L}_1^{\text{loc}}([0, \infty \mapsto \mathcal{W}_c)$ there are only a finite number of control update times (i.e. $\tau_{k+1} = \infty$ for some k and thus $\tau_{k+2}, \tau_{k+3}, \ldots$ do not exist). In this case we have that the corresponding state trajectory denoted by $x_{x_0,w}$ lies inside $\mathcal{B}$ for all times $t > \tau_k + T_s$ in the non-uniform mechanism and for all control check times $jT_s > \tau_k$ in case of

[2]This is merely a technical condition to make the following exposition more compact and clear. This is not a restrictive condition.

the uniform mechanism. Hence, for state trajectories where this phenomenon occurs we already have some ultimate boundedness properties. We introduce the notation $\mathcal{S}(x_0, w)$ in this context as the index set corresponding to all *finite* control update times for initial state x_0 and disturbance signal w. The notation $\mathcal{S}'(x_0, w)$ is the index set corresponding to all control update times τ_κ that are not only finite themselves, but also the next control update time $\tau_{\kappa+1}$ is finite. In the above situation with $\tau_k < \infty$ and $\tau_{k+1} = \infty$, $\mathcal{S}(x_0, w) = \{0, \ldots, k\}$ and $\mathcal{S}'(x_0, w) = \{0, \ldots, k-1\}$.

With regard to practical implementation, it has to be observed that the uniform mechanism is easier to implement, although it is more difficult to analyze, as we will see.

As already mentioned, the control objective is a "practical stabilization problem" in the sense of controlling the state towards a region Ω close to the origin and keeping it there.

Problem 5.1 *Let a desired ultimate bound $\Omega \subset \mathbb{R}^n$ containing 0 in the interior be given. Construct F and $\mathcal{B}$ such that the system* (5.12) *with the control update times given by either* (5.13) *or* (5.14) *is UB to Ω.*

For the moment, we ignore the transient behavior of the event-driven system. The reason is that this is easily inherited from properties of the discrete-time linear system with the fixed sample time T_s (cf. (5.15) below). We will return to this issue later in section 5.10 in which it is explained how to tune the controller to get a satisfactory ultimate bound Ω and convergence rate towards Ω.

5.5 Approach

Problem 5.1 will be solved in two stages as is typical for sampled-data systems. First properties on UB to Ω are obtained for the event-driven system (5.12) *on the control update times* only. Next bounds on the intersample behavior (see section 5.8 below) will be derived that enlarge Ω to $\tilde{\Omega}$ such that the ultimate bound $\tilde{\Omega}$ is guaranteed for all (continuous) times t.

We first introduce the formal definitions of robust positive invariance and ultimate boundedness "on the control update times" for the system (5.12) together with a particular event-triggering mechanism.

Definition 5.3 *Consider the system* (5.12) *with either* (5.13) *or* (5.14) *as the event-triggering mechanism.*

- *For this system the set $\Omega \subseteq \mathbb{R}^n$ is called* robustly positively invariant (RPI) on the control update times *to the set Ω for disturbances in $\mathcal{W}_c$, if for any initial state x_0 and $w \in \mathcal{L}_1^{\text{loc}}([0,\infty) \mapsto \mathcal{W}_c)$ the corresponding state trajectory $x_{x_0,w}$ of the system has the property that $x_{x_0,w}(\tau_k) \in \Omega$ for some $k \in \mathcal{S}'(x_0,w)$ implies $x_{x_0,w}(\tau_{k+1}) \in \Omega$.*

- *This system is* ultimately bounded (UB) on the control update times *for disturbances in $\mathcal{W}_c$, if for any initial condition x_0 there exists a $K(x_0)$ such that for any $w \in \mathcal{L}_1^{\text{loc}}([0,\infty) \mapsto \mathcal{W}_c)$ the corresponding state trajectory $x_{x_0,w}$ satisfies for all $k \in \mathcal{S}(x_0,w)$, $k \geq K(x_0)$ that $x_{x_0,w}(\tau_k) \in \Omega$.*

Note that we only impose robust positive invariance or UB-related conditions on the *finite* control update times and not for time instants beyond. However, as noted in the previous section, for state trajectories x with $\mathcal{S}(x_0,w) = \{0,\ldots,k\}$ a finite collection, it holds that $x_{x_0,w}(t) \in \mathcal{B}$ for $t > \tau_k + T_s$ in the non-uniform mechanism and $x_{x_0,w}(jT_s) \in \mathcal{B}$ for all $jT_s > \tau_k$ in case of the uniform mechanism. Hence, we already have some ultimate boundedness properties with respect to the set $\mathcal{B}$ in this situation.

In the analysis the discrete-time system

$$x_{k+1}^d = (A+BF)x_k^d + w_k^d = A_{cl}x_k^d + w_k^d \tag{5.15}$$

with

$$\begin{aligned} A &:= e^{A_c T_s} \\ B &:= \textstyle\int_0^{T_s} e^{A_c\theta} d\theta B_c \\ w_k^d &:= \textstyle\int_{\tau_k}^{\tau_{k+1}} e^{A_c(\tau_{k+1}-\theta)} E_c w(\theta) d\theta \\ A_{cl} &:= A + BF \end{aligned} \tag{5.16}$$

will play an important role. Indeed, for both the uniform and non-uniform sampling case, the system behaves away from the set $\mathcal{B}$ (at the control update times) as (5.15). We use the shorthand notation $x_{x_0,w}(\tau_k) = x_k^d$ here. This system is only representing the system (5.12) at the control update times, when $\tau_{k+1} = \tau_k + T_s$. The bounds on $w(t)$ via $\mathcal{W}_c$ are transformed into bounds on w_k^d given by

$$\mathcal{W}_d := \{\int_0^{T_s} e^{A_c(T_s-\theta)} E_c w(\theta) d\theta \mid w \in \mathcal{L}_1([0,T_s] \mapsto \mathcal{W}_c)\}. \tag{5.17}$$

Since $\mathcal{W}_c$ is convex, compact and contains 0, $\mathcal{W}_d$ is convex, compact and contains 0.

5.6 Main results for non-uniform mechanism

The first theorem below states that ultimate bounds for the *linear discrete-time system* (5.15) can be used to find ultimate bounds for the *event-driven system* (5.12) on the control update times $\{\tau_k\}_k$ with non-uniform sampling (5.13).

Theorem 1 *Consider the system* (5.12)-(5.13) *with $\mathcal{W}_c$ a closed, convex set containing* 0, *F given and $\mathcal{B}$ an open set containing the origin. Let $\mathcal{W}_d$ be given by* (5.17).

1. *If Ω is a RPI set for the* linear discrete-time system (5.15) *with disturbances in $\mathcal{W}_d$ and cl$\mathcal{B} \subseteq \Omega$, then Ω is a RPI set for the* event-driven system (5.12)-(5.13) on the control update times *for disturbances in $\mathcal{W}_c$.*

2. *If the* linear discrete-time system (5.15) *with disturbances in $\mathcal{W}_d$ is UB to the RPI set Ω and cl$\mathcal{B} \subseteq \Omega$, then the* event-driven system (5.12)-(5.13) on the control update times *is UB to Ω for disturbances in $\mathcal{W}_c$.*

Proof 1) Let an arbitrary initial state x_0 and $w \in \mathcal{L}_1^{\text{loc}}([0,\infty) \mapsto \mathcal{W}_c)$ be given and consider the corresponding state trajectory $x_{x_0,w}$ of the system (5.12)-(5.13) and assume that $x_{x_0,w}(\tau_k) \in \Omega$ for some $k \in \mathcal{S}'(x_0, w)$. Since τ_{k+1} is finite as well, there are two possibilities:

- $\tau_{k+1} = \tau_k + T_s$. This means that the update of the state over the interval $[\tau_k, \tau_{k+1}]$ is governed by (5.15) for some $w_k^d \in \mathcal{W}_d$. Since Ω is a RPI set for (5.15), this means that $x_{x_0,w}(\tau_{k+1}) \in \Omega$ (irrespective of the realization of the disturbance).

- $\tau_{k+1} \neq \tau_k + T_s$. Note that for $k > 0$ it holds that $\tau_{k+1} > \tau_k + T_s$. Only for $k = 0$, it may hold that $\tau_1 \leq \tau_0 + T_s$. According to (5.13) this implies that $x_{x_0,w}(\tau_{k+1}) \in \partial\mathcal{B} \subset$cl$\mathcal{B}$. Since cl$\mathcal{B} \subseteq \Omega$, it holds that $x_{x_0,w}(\tau_{k+1}) \in \Omega$.

This proves that Ω is RPI for (5.12)-(5.13) on the control update times for disturbances in $\mathcal{W}_c$.

2) Consider an initial state x_0. If $x_0 \in \Omega$, then due to the RPI property of Ω on the control update times as proven in the first part of the proof, the state trajectory $x_{x_0,w}$ stays within Ω on the control update times (irrespective of the disturbance signal $w \in \mathcal{L}_1^{\text{loc}}([0,\infty) \mapsto \mathcal{W}_c)$), i.e. $x_{x_0,w}(\tau_k) \in \Omega$ for $k \in \mathcal{S}(x_0, w)$. Hence, one can take $K(x_0) = 0$ in definition 5.3. Suppose $x_0 \notin \Omega$. Since (5.15) is UB to Ω for disturbances

in $\mathcal{W}_d$, there exists a time $K(x_0)$ such that any discrete-time trajectory x^d with initial condition $x_0^d = x_0$ of (5.15) satisfies $x_k^d \in \Omega$ for $k \geq K(x_0)$ (irrespective of the disturbance signal $w \in \mathcal{W}_d^\infty$). We claim that this $K(x_0)$ satisfies also definition 5.3 for x_0.

Indeed, let $w \in \mathcal{L}_1^{\text{loc}}([0,\infty) \mapsto \mathcal{W}_c)$ and consider the trajectory $x_{x_0,w}$ of (5.12)-(5.13) for initial condition $x(0) = x_0$ and disturbance signal w. Let $\bar{k} \in \mathcal{S}(x_0, w)$ satisfy $\bar{k} \geq K(x_0)$. We proceed by contradiction. Assume that $x_{x_0,w}(\tau_k) \notin \Omega$ for $k = 0, \ldots, K(x_0)$. Since $x_{x_0,w}(\tau_k) \notin \Omega$ for $k = 0, \ldots, \bar{k}$, and thus $x_{x_0,w}(\tau_k) \notin \mathcal{B}$ for $k = 0, \ldots, \bar{k}$, it holds that $\tau_{k+1} = \tau_k + T_s$ and $x_{x_0,w}(\tau_{k+1})$ and $x_{x_0,w}(\tau_k)$ are related through (5.15) for some $w_k^d \in \mathcal{W}_d$ for all $k = 0, \ldots, \bar{k} - 1$. Hence, $x_{x_0,w}(\tau_k) = x_k^d$ for $k = 0, \ldots, \bar{k}$. However, $x_{K(x_0)}^d \in \Omega$ and $\bar{k} \geq K(x_0)$. We reached a contradiction. Hence, there is a $k \in \{0, \ldots, K(x_0)\}$, say $\tilde{k}$ such that $x_{x_0,w}(\tau_{\tilde{k}}) \in \Omega$. Since Ω is RPI for (5.12)-(5.13) on the control update times, we have $x_{x_0,w}(\tau_k) \in \Omega$ for all $k \geq \tilde{k}$ and $k \in \mathcal{S}(x_0, w)$. As $K(x_0) \geq \tilde{k}$, this completes the proof of statement 2. □

5.7 Main results for the uniform mechanism

As mentioned before, the non-uniform update scheme is hard to implement in practice. Uniform sampling might be more relevant from a practical point of view. However, in contrast to non-uniform sampling the properties of the discrete-time linear system *do not* transfer to the event-driven system in this case. As we will see, we will need a discrete-time piecewise linear (PWL) model (see e.g. [37, 81]) to analyze the event-driven systems using uniform sampling. We will present two approaches to this problem. A first PWL model uses $(x^T(\tau_k), u^T(\tau_{k-1}))^T$ as state variable, while the second PWL model only uses $x(\tau_k)$. This implies that the first model has a higher $(n+m)$-dimensional state variable than the second (n-dimensional), but as we will see next, it only needs two linear submodels, while the second PWL model might need much more.

5.7.1 A bimodal higher order piecewise linear system

To derive results on ultimate boundedness on the control update times for the event-driven system with the uniform mechanism, we will embed the control update times $\{\tau_k \mid k \in \mathcal{S}(x_0, w)\}$ in its superset $\{jT_s \mid j \in \mathbb{N}\}$. From (5.12)-(5.14) together with the discretization (5.16) it can be observed that the behavior of the system (5.12) and

(5.14) on the *control check times* $\{jT_s \mid j \in \mathbb{N}\}$ can be included in

$$\begin{pmatrix} x_{k+1}^d \\ u_k^d \end{pmatrix} = \begin{cases} \begin{pmatrix} A+BF & 0 \\ F & 0 \end{pmatrix} \begin{pmatrix} x_k^d \\ u_{k-1}^d \end{pmatrix} + \begin{pmatrix} I \\ 0 \end{pmatrix} w_k^d, & \text{if } x_k^d \notin \mathcal{B} \\ \begin{pmatrix} A & B \\ 0 & I \end{pmatrix} \begin{pmatrix} x_k^d \\ u_{k-1}^d \end{pmatrix} + \begin{pmatrix} I \\ 0 \end{pmatrix} w_k^d, & \text{if } x_k^d \in \mathcal{B} \end{cases} \tag{5.18}$$

for $w_k^d \in \mathcal{W}_d$. Note that the fact $\tau_0 = 0$, i.e. at the initial time a control update $u_0 = Fx_0$ is performed, can be included by considering initial states in the set $X_0 := \{(x_0^T, u_{-1}^{d\,T})^T \mid u_{-1}^d = Fx_0\}$.

We need the following definition.

Definition 5.4 *Let Γ be a subset of $\mathbb{R}^n \times \mathbb{R}^m$. The projection $\Pi_n(\Gamma)$ of Γ on the first n components of the vector space $\mathbb{R}^n \times \mathbb{R}^m$ is defined as $\{x \in \mathbb{R}^n \mid \exists u \in \mathbb{R}^m$ such that $(x^T, u^T)^T \in \Gamma\}$*

As $\{\tau_k \mid k \in \mathcal{S}(x_0, w)\} \subseteq \{jT_s \mid j \in \mathbb{N}\}$ for any x_0 and disturbance signal w, we will formulate a result on ultimate boundedness on the control *check* times, which can be defined analogously as for the control update times in definition 5.3.

Theorem 2 *Consider the system* (5.12) *and* (5.14) *with $\mathcal{W}_c$ a closed, convex set containing* 0*, F given and $\mathcal{B}$ an open set containing the origin. Let $\mathcal{W}_d$ be given by* (5.17)*. If the* piecewise linear discrete-time system (5.18) *with disturbances in $\mathcal{W}_d$ is UB for initial states in X_0 to the set Γ, then the* event-driven system (5.12) *and* (5.14) on the control check times *is UB to $\Pi_n(\Gamma)$ for disturbances in $\mathcal{W}_c$.*

Proof As system (5.12) and (5.14) and (5.18) coincide on the control check times and (5.18) is UB to Ω, we have that for any x_0 there exists a $K(x_0, u_{-1}^d)$ with $u_{-1}^d = Fx_0$ such that $(x_k^{d\,T}, u_{k-1}^{d\,T})^T \in \Gamma$ for all $k \geq K(x_0, u_{-1}^d)$. Since u_{-1}^d is a function of x_0, it holds that there is a $\tilde{K}(x_0) := K(x_0, Fx_0)$ such that $x_{x_0,w}(\tau_k) \in \Pi_n(\Gamma)$ for all $k \geq \tilde{K}(x_0)$.

5.7.2 A multi-modal lower order piecewise linear system

As already mentioned, the above approach leads to a piecewise linear system with an $(m+n)$-dimensional state vector. As this might be prohibitive for numerical tools computing ultimate bounds for PWL systems (when the number of control inputs m is large), an alternative approach is presented in this section.

5.7.2.1 The unperturbed case

Consider (5.12) on the *control check times*, which can be described by

$$\begin{aligned} x_{k+1}^d &= Ax_k^d + Bu_k^d \\ u_k^d &= \begin{cases} Fx_k^d, & \text{if } x_k^d \notin \mathcal{B} \\ u_{k-1}^d, & \text{if } x_k^d \in \mathcal{B} \end{cases} \end{aligned} \tag{5.19}$$

with $x_0^d := x_0$ and $u_{-1}^d = Fx_0$. We make the following observation. When the state x_k^d is outside $\mathcal{B}$ the update matrices in (5.18) do not depend on u_{k-1}^d and hence, the information on u_{k-1}^d is not necessary. Only in case $x_k^d \in \mathcal{B}$ the previous control value has to be known as it is going to be held for at least one, but possible multiple check times. However, we can explicitly compute this update relation depending on how many check times the control value is held. This update relation will map the state just before entering $\mathcal{B}$ (at a control update time) to the state just after leaving $\mathcal{B}$ again (at the next control update time). It will turn out that in this way a piecewise linear (PWL) model is obtained in which we abstract away from the number of discrete-time steps that the system is inside $\mathcal{B}$. Using this *PWL system* properties related to UB can again be translated to the original system (5.12) and (5.14) on *the control update times*. The advantage with respect to (5.18) is that we only have an n-dimensional state vector now.

To define the map h_p for the different periods of time (denoted by p) that the state stays in $\mathcal{B}$, we consider first the case $p = 0$, i.e. $x_k^d \notin \mathcal{B}$ and $x_{k+1}^d \notin \mathcal{B}$ the system update matrix is given by

$$x_{k+1}^d := h_0(x_k^d) = (A + BF)x_k^d. \tag{5.20}$$

For $p = 1$ we assume that $x_k^d \notin \mathcal{B}$, $x_{k+1}^d \in \mathcal{B}$ and then $x_{k+2}^d \notin \mathcal{B}$. The function h_1 defines the mapping from x_k^d to x_{k+2}^d in this case. This update of the state variable is given by $x_{k+i}^d = Ax_{k+i-1}^d + Bu_{k+i-1}^d$, $i = 1, 2$ with $u_{k+1}^d = u_k^d = Fx_k^d$ (since the control value is held). Hence,

$$x_{k+2}^d \quad = \quad h_1(x_k^d) := [A(A + BF) + BF]x_k^d \tag{5.21}$$

Similarly, suppose we stay p steps in $\mathcal{B}$ before leaving $\mathcal{B}$ again (i.e. $x_k^d \notin \mathcal{B}$, then $x_{k+1}^d \in \mathcal{B}, x_{k+2}^d \in \mathcal{B}$, ..., $x_{k+p}^d \in \mathcal{B}$ and then $x_{k+p+1}^d \notin \mathcal{B}$). We obtain the function h_p that maps x_k^d to x_{k+p+1}^d as follows by using repetitively $x_{k+i}^d = Ax_{k+i-1}^d + Bu_{k+i-1}^d$, for $i = 1, ..., p + 1$. Since $u_{k+p}^d = u_{k+p-1}^d = \ldots = u_k^d = Fx_k^d$, we can express

x^d_{k+p+1} as a function of x^d_k:

$$x^d_{k+p+1} \quad = \quad h_p(x^d_k) := \{A^p(A+BF) + [A^{p-1} + A^{p-2} + \ldots + I]BF\}x^d_k. \tag{5.22}$$

Now that the maps h_p are defined, the regions D_p must be determined in which the map h_p is active. For $p = 0$, this is straightforward as $D_0 := \mathcal{B}^c$, which denotes the complement of $\mathcal{B}$. For $p > 0$ D_p is given by those $x^d_k \notin \mathcal{B}$ that satisfies $x^d_{k+1} \in \mathcal{B}$, $x^d_{k+2} \in \mathcal{B}$, ..., $x^d_{k+p} \in \mathcal{B}$ and $x^d_{k+p+1} \notin \mathcal{B}$. Hence, for $p = 0, 1, 2, \ldots$ we have

$$D_p := \{x \notin \mathcal{B} \mid h_j(x) \in \mathcal{B} \text{ for } j = 0, 1, \ldots, p-1 \text{ and } h_p(x) \notin \mathcal{B}\}. \tag{5.23}$$

We also define the set of states that remain inside $\mathcal{B}$ forever after entering it from outside $\mathcal{B}$.

$$D_\infty := \{x \notin \mathcal{B} \mid h_j(x) \in \mathcal{B} \text{ for all } j = 0, 1, \ldots\}. \tag{5.24}$$

Note that $D_i \cap D_j = \emptyset$ if $i \neq j$.

Finally, we introduce the set $R_\mathcal{B}$ which contains all possible values of x^d_k outside $\mathcal{B}$, that can reach $\mathcal{B}$ within one discrete time-step:

$$R_\mathcal{B} := \{x \notin \mathcal{B} \mid h_0(x) \in \mathcal{B}\}$$

. With these definitions, it holds that

$$\mathbb{R}^n = \mathcal{B} \cup R_\mathcal{B} \cup D_0 \text{ and } R_\mathcal{B} = D_\infty \cup \bigcup_{i=1}^{\infty} D_i. \tag{5.25}$$

To obtain a finite representation of the piecewise linear system, we need the existence of a $p_{\max}$ such that

$$R_\mathcal{B} = D_\infty \cup \bigcup_{i=1}^{p_{\max}} D_i \tag{5.26}$$

in which $p_{\max}$ is the maximal (finite) number of discrete steps that the system (5.19) can stay inside $\mathcal{B}$ after entering it from outside $\mathcal{B}$.

Deriving conditions for which the existence of such a finite $p_{\max}$ is guaranteed is an open issue. One of the complications is for instance that $D_i = \emptyset$ does not necessarily imply that $D_{i+1} = \emptyset$. Also the computation of D_∞ is not straightforward. A computational approach can be obtained by increasing $p_{\max}$ until the right-hand side of (5.26) is equal to the left-hand side. However, still analytical results proving the existence of a finite $p_{\max}$ and possibly an upper bound for it, would be very beneficial. One such condition is formulated in section 5.7.2.2.

The iteration parameter k related to the *control check times* kT_s is replaced by a new "discrete-time variable" l corresponding to the *control update times* τ_l after abstracting away from the motion of the system's state inside $\mathcal{B}$. Therefore, we replace $x^d_{k+p+1} = h_p(x^d_k)$ by $x^d_{l+1} = h_p(x^d_l)$ and obtain the piecewise linear system $x^d_{l+1} = f_{\text{PWL}}(x^d_l)$ with

$$x^d_{l+1} = \begin{cases} h_p(x^d_l), & \text{when } x^d_l \in D_p \\ 0, & \text{when } x^d_l \in D_\infty \cup \mathcal{B}. \end{cases} \tag{5.27}$$

Some observations on the PWL system (5.27) are in order.

- The dynamics of (5.19) and (5.27) coincide on $\mathcal{B}^c \setminus D_\infty = \bigcup_{i=0}^{p_{\max}} D_i$ in the sense that $x_{x_0}(\tau_{l+1}) = x^d_{l+1} = f_{\text{PWL}}(x^d_l) = f_{\text{PWL}}(x_{x_0}(\tau_l))$ for $x^d_l = x_{x_0}(\tau_l) \in \mathcal{B}^c \setminus D_\infty$, where x_{x_0} denotes the solution of the event-driven system (5.12) and (5.14) for initial condition $x(0) = x_0$ and $\tau_l \in \mathcal{S}'(x_0)$ a control update time. Moreover, $x^d_l = x_{x_0}(\tau_l) \in D_\infty$ implies that $\tau_{l+1} = \infty$.

- On $D_\infty \cup \mathcal{B}$ the piecewise linear model was completed by adding dynamics to the system for the case when $x^d_l \in D_\infty$ and $x^d_l \in \mathcal{B}$. As will be proven below, it is not important how the dynamics are chosen exactly on these sets as long as they do not map outside $\mathcal{B}$.

- A set D_p is in general not convex. It might even not be connected. See, the second example in section 5.11.

We state now the main result of this section.

Theorem 3 *Consider system* (5.12) *and* (5.14) without *disturbances and $\mathcal{B}$ is an open set containing the origin. Assume that there exists a $p_{\max} < \infty$ such that* (5.26) *holds. Let $\mathcal{W}_d$ be given by* (5.17)*. If the PWL system* (5.27) *is UB to the positively invariant set Ω and $\mathcal{B} \subseteq \Omega$, then the* event-driven system (5.12) *and* (5.14) *is UB to* Ω on the control check times.

Proof The system (5.12) and (5.14) on the control check times is described by (5.19). Therefore, we consider solutions in terms of trajectories x^d of (5.19) in this proof.

If $x^d_0 \in \Omega$ then we either have that the state trajectory of (5.19) satisfies $x^d_k \in \Omega$ for all $k = 0, 1, 2, \ldots$ (which is in accordance with the properties of the theorem) or the state trajectory leaves Ω for some control check time. Hence, without loss of generality we can consider the case that there exists a k_0 (take the smallest) for which $x^d_{k_0} \notin \Omega$

and thus $x^d_{k_0} \in \mathcal{B}^c = D_\infty \cup \bigcup_{i=0}^{p_{\max}} D_i$ because $\mathcal{B} \subseteq \Omega$. Observe that the dynamics of (5.19) and (5.27) coincide on $\bigcup_{i=0}^{p_{\max}} D_i$ (modulo the motion inside $\mathcal{B}$, which lies in Ω and therefore is not affecting the UB to Ω). Hence, since $x^d_{k_0} \in D_\infty \cup \bigcup_{i=0}^{p_{\max}} D_i$, the system (5.19) follows the dynamics of the PWL system (5.27) (modulo the motion inside $\mathcal{B}$) for $k \geq k_0$ until D_∞ is reached - if ever (say at $k_1 \geq k_0$ with k_1 possibly equal to ∞). If D_∞ is reached, the state x^d_k of (5.19) stays inside $\mathcal{B} \subseteq \Omega$ for all $k > k_1$ by definition of D_∞. Hence, on the discrete-time interval $[k_0, k_0 + 1, \ldots, k_1)$ the state of system (5.19) follows the motion of (5.27) and hence, the inheritance of the UB property follows. □

Remark 4 *The theorem also holds for $p_{\max} = \infty$. However, the use of theorem in practice is lost due to the infinite character of the piecewise linear system.*

Note that the larger p is, the more event times we are not updating the control value and thus we are not using the CPU for performing control computations. So, the larger $p_{\max}$ the more we can potentially save on processor load, but the complexer (the more regions) the resulting PWL model will be for the performance analysis. Advantageously, the computation of the ultimate bounds is performed off-line.

5.7.2.2 Finite PWL representations

In this section we present a sufficient condition that guarantees the existence of a finite PWL representation (5.27) (i.e. the existence of a finite $p_{\max}$ such that (5.26) holds).

Theorem 5 *Consider system* (5.19). *Assume that all the eigenvalues of the matrix A lie outside the closed unit circle of the complex half plane and $A + BF$ does not have an eigenvalue* 1 *(which is typically the case as $A + BF$ is chosen such that all eigenvalues are inside the open unit circle). Then* (5.26) *holds for a finite $p_{\max}$ and $D_\infty = \emptyset$.*

Proof We need two algebraic results in the proof, that will be established next.

- Since A has all its eigenvalues outside the closed unit circle, A^{-1} is Schur (i.e. all eigenvalues inside the unit circle). This implies that there is a positive definite matrix P such that $(A^{-1})^T P A^{-1} - P < 0$. Premultiplying the latter inequality by A^T and postmultiplying by A and noting that A is invertible

yields $P - A^T PA < 0$. Hence, there exists a $\gamma > 1$ such that

$$A^T PA > \gamma P \tag{5.28}$$

- Next we prove that the matrix $(A+BF)-(I-A)^{-1}BF$ is invertible. Suppose that $(A+BF)z = (I-A)^{-1}BFz$. This implies $A(I-A-BF)z = 0$. Since A is invertible this yields $(A+BF)z = z$. As $A+BF$ does not have an eigenvalue 1, this give $z = 0$ and hence, the invertibility of $(A + BF) - (I - A)^{-1}BF$ is proven.

To finish the proof, we recall that $p_{\max}$ is the maximal number of discrete steps that the system (5.19) can stay inside $\mathcal{B}$ after entering it from outside $\mathcal{B}$. Let x_0 be the last state outside $\mathcal{B}$ and $x_1 := (A + BF)x_0$ the first state inside $\mathcal{B}$. Inside $\mathcal{B}$ the state is governed by

$$x_{k+1} = Ax_k + BFx_0. \tag{5.29}$$

as the input is held at the value Fx_0. For shortness of notation, we omit superscript d in this proof. The system (5.29) has an (unstable) equilibrium at $x_{eq} := (I-A)^{-1}BFx_0$. If we define $\Delta x_k := x_k - x_{eq}$, $k = 0, 1, 2, \ldots$ then we can observe that $\Delta x_{k+1} = A\Delta x_k$. Together with (5.28) this yields $\Delta x_k^T P \Delta x_k > \gamma^{k-1} \Delta x_1^T P \Delta x_1$. Note that $\Delta x_1 = [(A + BF) - (I - A)^{-1}BF]x_0$. Since $\mathcal{B}$ contains 0 in its interior, $x_0 \notin \mathcal{B}$, $(A + BF) - (I - A)^{-1}BF$ is invertible and P is positive definite, it holds that $\Delta x_1^T P \Delta x_1 \geq \mu$ for some $\mu > 0$. Hence, $\Delta x_k P \Delta x_k \geq \gamma^{k-1}\mu$ (independent of x_0). The latter inequality indicates that x_k will move arbitrarily far away from $x_1 \in \mathcal{B}$ for sufficiently large k. Indeed,

$$\begin{aligned}
(x_k - x_1)^T P(x_k - x_1) &= (\Delta x_k - \Delta x_1)^T P(\Delta x_k - \Delta x_1) \\
&= \|P^{\frac{1}{2}}(\Delta x_k - \Delta x_1)\|^2 \\
&\geq (\|P^{\frac{1}{2}}\Delta x_k\| - \|P^{\frac{1}{2}}\Delta x_1\|)^2 \\
&> (\sqrt{\gamma^{k-1}} - 1)^2 \|P^{\frac{1}{2}}\Delta x_1\|^2 \geq \\
&\geq (\sqrt{\gamma^{k-1}} - 1)^2 \mu
\end{aligned}$$

Since $\mathcal{B}$ is bounded the expression $\delta := \sup\{(x - x_1)^T P(x - x_1) \mid x \in \mathcal{B},\ x_1 \in \mathcal{B}\}$ is finite. Hence, if k is large enough to satisfy $(\sqrt{\gamma^{k-1}} - 1)^2\mu > \delta$, it follows that x_k must be outside $\mathcal{B}$ (as x_1 lies inside $\mathcal{B}$). This completes the proof. □

An upper bound on $p_{\max}$ follows from the proof above.

Corollary 6 *Consider system* (5.19)*. Assume that all the eigenvalues of the matrix* A *lie outside the closed unit circle of the complex half plane and* $A + BF$ *does not have an eigenvalue* 1*.*

- *Let* $P > 0$ *be a solution to* $A^T P A > \gamma P$ *for a* $\gamma > 1$*, which is known to exist.*
- $\mu := \min_{z \notin \mathcal{B}} z^T[(A+BF)-(I-A)^{-1}BF]^T P[(A+BF)-(I-A)^{-1}BF]z > 0$
- $\delta := \sup\{(x - x_1)^T P(x - x_1) \mid x \in \mathcal{B},\ x_1 \in \mathcal{B}\}$

Let $k_{\min}$ *be the smallest integer* k *that satisfies* $(\sqrt{\gamma^{k-1}} - 1)^2 \mu > \delta$*. Then* $p_{\max} \leq k_{\min}$

5.7.2.3 The perturbed case

In this subsection we briefly indicate how the derivation given above needs to be modified in order to include additive disturbance in the event-driven system (5.12) and (5.14). At the control check times the trajectory of the system (5.12) and (5.14) is described by the discrete-time system

$$\begin{aligned} x_{k+1}^d &= Ax_k^d + Bu_k^d + w_k^d \\ u_k^d &= \begin{cases} Fx_k^d & \text{if } x_k^d \notin \mathcal{B} \\ u_{k-1}^d & \text{if } x_k^d \in \mathcal{B}. \end{cases} \end{aligned} \tag{5.30}$$

for some realization of the disturbance $w_k^d \in \mathcal{W}_d$, $k = 0, 1, 2, \ldots$. In this case we will also compute a PWL system, but now the mappings h_p will depend not only on the state x_k^d but also on the disturbance sequence $(w_k^d, w_{k+1}^d, \ldots, w_{k+p}^d)$. Suppose the state trajectory stays p steps in $\mathcal{B}$ before leaving $\mathcal{B}$ again (i.e. $x_k \notin \mathcal{B}$, then $x_{k+1}^d \in \mathcal{B}$, $x_{k+2}^d \in \mathcal{B}$, ..., $x_{k+p}^d \in \mathcal{B}$ and then $x_{k+p+1}^d \notin \mathcal{B}$). We obtain the function h_p that maps x_k^d to x_{k+p+1}^d similarly as in section 5.7.2.1

$$\begin{aligned} x_{k+p+1}^d &= h_p(x_k^d, w_{k+p}^d, \ldots, w_k^d) \\ &= Ah_{p-1}(x_k^d, w_{k+p-1}^d, \ldots, w_k^d) + BFx_k^d + w_{k+p}^d \\ &= \{A^p(A + BF) + [A^{p-1} + A^{p-2} + \ldots + I]BF\}x_k^d \\ &\quad + [A^p w_k^d + A^{p-1} w_{k+1}^d + \ldots + w_{k+p}^d]. \end{aligned} \tag{5.31}$$

One can observe that the dynamics depend on different sizes of the disturbance sequence $(w_k^d, w_{k+1}^d, \ldots, w_{k+p}^d) \in \underbrace{\mathcal{W}_d \times \ldots \times \mathcal{W}_d}_{p+1 \text{ times}} =: \mathcal{W}_d^{p+1}$. In this sense we could

describe the system by using an "embedding" in the product space $\mathbb{R}^n \times l_\infty^n$, where l_∞^n denotes the space of (infinite) sequences $(w_0^d, w_1^d, w_2^d, \ldots)$ that are bounded in the sense that $\sup_{k\in\mathbb{N}} \|w_k^d\| < \infty$. Indeed, all the maps h_p can be reconsidered as having arguments in $\mathbb{R}^n \times l_\infty^n$ by defining for $p = 0, 1, 2, \ldots$

$$x_{k+p}^d = H_p(x_k^d, w_k^d) = h_p(x_k^d, w_{k+p}^d, \ldots, w_k^d), \tag{5.32}$$

for $(x_k^d, w_k^d) \in \mathbb{R}^n \times l_\infty^n$, where $w_k^d = (w_k^d, \ldots, w_{k+p-1}^d, w_{k+p}^d, \ldots)$. Each map H_p is valid on regions D_p that can be determined as

$$\begin{aligned} D_p \quad := \quad & \{(x_k^d, w_k^d) \in \mathcal{B}^c \times \mathcal{W}_d^\infty \mid H_j(x_k^d, w_k^d) \in \mathcal{B} \text{ for } j = 0, 1, \ldots, p-1 \\ & \text{and } H_p(x_k^d, w_k^d) \notin \mathcal{B}\}. \end{aligned} \tag{5.33}$$

In a similar manner as for the unperturbed case, we also define the set of states and disturbance sequences that remain inside $\mathcal{B}$ forever after entering it from outside $\mathcal{B}$.

$$\begin{aligned} D_\infty := \{(x_k^d, w_k^d) \in \mathcal{B}^c \times \mathcal{W}_d^\infty \mid \quad & H_j(x_k^d, w_k^d) \in \mathcal{B} \\ & \text{for all } j = 0, 1, 2, \ldots\}. \end{aligned} \tag{5.34}$$

Note that $D_i \cap D_j = \emptyset$ if $i \neq j$. Moreover, observe that in this case the "switching" of the dynamics is dependent on the disturbance input as well and not solely on the state as in the unperturbed case.

Finally, we introduce the set $R_\mathcal{B}$ which contains all possible values (x_k^d, w_k^d) in $\mathcal{B} \times l_\infty^n$ for which x_k^d can be reached within one discrete time-step starting from a state x_{k-1}^d outside $\mathcal{B}$ by disturbance input w_{k-1}^d:

$$R_\mathcal{B} := \{(x_k^d, w_k^d) \in \mathcal{B}^c \times \mathcal{W}_d^\infty \mid H_0(x_k^d, w_k^d) \in \mathcal{B}\}. \tag{5.35}$$

Similarly to the unperturbed case, (5.25) holds. However, it does not hold in $\mathbb{R}^n$, but in the embedding space $\mathbb{R}^n \times \mathcal{W}_d^\infty$. Moreover, to obtain a finite representation of the PWL system, we need the existence of a $p_{\max}$ such that (5.26) holds, where $p_{\max}$ is the maximal (finite) number of discrete steps that the system (5.19) can stay inside $\mathcal{B}$ after entering it from outside $\mathcal{B}$ (for a particular disturbance realization). In the perturbed case, there are two reasons for the "infinite representation" of the PWL system; first of all the number of regions can be infinite (as in the unperturbed case), but also the length of the disturbances sequence determining the update from x_k^d to x_{k+p+1}^d can be infinite. Hence, the existence of a finite $p_{\max}$ leads on one hand to a finite number of regions of the PWL system and on the other implies that the infinitely dimensional

space $\mathbb{R}^n \times l_\infty^n$ can be replaced by $\mathbb{R}^n \times (\mathbb{R}^n)^{p_{\max}+1}$. Indeed, if we abstract away from the motion inside $\mathcal{B}$ and replace the iteration parameter k corresponding to the control check times by the new discrete-time variable l corresponding to the control update times, we obtain the PWL system $x_{l+1}^d = f_{\text{PWL}}(x_l^d, w_l^d)$ with

$$x_{l+1}^d = \begin{cases} H_p(x_l^d, w_l^d), & \text{when } (x_l^d, w_l^d) \in D_p, p = 0, 1, \ldots, p_{\max} \\ 0, & \text{when } (x_l^d, w_l^d) \notin \bigcup_{p=0}^{p_{\max}} D_p \end{cases} \tag{5.36}$$

with $w_l^d \in \mathcal{W}_d^{p_{\max}+1}$. Note that there is a slight abuse of notation in (5.36) as we replaced $w_k^d \in \mathcal{W}_d^\infty$ by $w_l^d \in \mathcal{W}_d^{p_{\max}+1}$ in both H_p and D_p.

A similar result as theorem 3 can be derived in this case as well.

5.8 Including intersample behavior

The above results only provide statements *on the control update or control check times*. The behavior of the system in between these control check/update times is not characterized. However, since at the control check/update times we obtain UB to a set Ω, we know that the state trajectories enter Ω in finite time. Using this observation, an ultimate bound *including* the intersample behavior of (5.12) together with (5.13) or (5.14) can be computed from

$$\begin{aligned} x_{x_0,w}(t) - x_{x_0,w}(\tau_k) &= [e^{A_c(t-\tau_k)} - I]x_{x_0,w}(\tau_k) + \int_{\tau_k}^t e^{A_c(t-\theta)} B_c u(\tau_k) d\theta \\ &\quad + \int_{\tau_k}^t e^{A_c(t-\theta)} E_c w(\theta) d\theta, \end{aligned} \tag{5.37}$$

where $t \in [\tau_k, \tau_{k+1})$.

In the non-uniform case we either have $x_{x_0,w}(t) \in \mathcal{B}$ or $t - \tau_k < T_s$ in (5.37). In the latter case using the boundedness of $\mathcal{W}_c$ we can easily see that

$$\|x_{x_0,w}(t) - x_{x_0,w}(\tau_k)\| \leq C T_s (\|x_{x_0,w}(\tau_k)\| + 1 + \|F\| \|x_{x_0,w}(\tau_k)\|) \tag{5.38}$$

for all $T_s \in [0, T_s^{\max}]$. The constant $C = C(A_c, B_c, E_c, T_s^{\max}, \mathcal{W}_c)$ depends on the system parameters, $A_c, B_c, E_c, \mathcal{W}_c$ and $T_s^{\max}$. Hence, if the system (5.15) is UB to a RPI set Ω with $\text{cl}\mathcal{B} \subseteq \Omega$ (as in theorem 1), then the event-driven system (5.12)-(5.13) is UB to the set $\tilde{\Omega} := \Omega \oplus B(0, \varepsilon)$ with $\varepsilon := \sup_{x \in \Omega} C T_s (\|x\| + 1 + \|F\| \|x\|)$ and $B(0, \varepsilon) := \{x \mid \|x\| \leq \varepsilon\}$.

In the uniform case a similar bound holds as in (5.38) with the minor modification

$$\|x_{x_0,w}(t) - x_{x_0,w}(jT_s)\| \leq CT_s(\|x_{x_0,w}(jT_s)\| + 1 + \|F\|\|x_{x_0,w}(\tau_k)\|), \quad (5.39)$$

where jT_s and τ_k are the largest control check time and largest control update time smaller than t, respectively. Two situations can occur: Either $\mathcal{S}(x_0, w)$ is a finite or an infinite set. In the latter case both $x_{x_0,w}(jT_s)$ and $x_{x_0,w}(\tau_k)$ lie ultimately in Ω, which yields a similar bound as in the uniform case.

In the former case, there is a $\bar{k}$ such that $x_{x_0,w}(kT_s) \in \mathcal{B}$ for $k \geq \bar{k}$, but it might be the case that $x_{x_0,w}(\tau_k)$ is outside Ω (it is just the last state at which a control update was performed). The only information on $x_{x_0,w}(\tau_k)$ is that it lies in $R_{\mathcal{B}}$ (in the unperturbed case) or there exists a w_k^d such that $(x_{x_0,w}(\tau_k), w_k^d) \in R_{\mathcal{B}}$ (the perturbed case). In case that $A + BF$ is invertible, the set $R_{\mathcal{B}}$ is bounded, which gives a bound on $\|x_{x_0,w}(\tau_k)\|$. Hence, using (5.39) a bound on the intersample behavior can be derived. Note that $D_\infty = \emptyset$ implies that this case is absent. Hence, if the conditions of theorem 5 are fulfilled, $D_\infty = \emptyset$ and this situation does not have to be considered.

Alternatively, in case there are physical reasons that the control inputs are restricted to a bounded set or if the method outlined in section 5.7.1 is used in which also ultimate boundedness of the u-variables is proven, a bound like $CT_s(\|x_k\| + 1)$ can be directly computed independent of the designed controller gain F.

5.9 Computational aspects

5.9.1 Computational aspects for the non-uniform case

In theorem 1 it was shown that properties of robust invariance of sets and UB for the discrete-time linear system (5.15) carry over to the event-driven system (5.12)-(5.13) on the control update times. As such it is of importance to be able to compute RPI sets and UB for discrete-time linear systems.

Definition 5.5 (Minimal Robustly Positively Invariant Set $\mathcal{F}_\infty$) *The set $\mathcal{F}_\infty$ is the minimal robustly positively invariant set of the discrete-time linear system* (5.15)*, if*

1. $0 \in \mathcal{F}_\infty$,
2. *$\mathcal{F}_\infty$ is robustly positively invariant of* (5.15) *with disturbances in $\mathcal{W}_d$,*
3. *and any other robustly positively invariant set F of* (5.15) *with disturbances in $\mathcal{W}_d$ satisfies $\mathcal{F}_\infty \subseteq \mathcal{F}$.*

It is well-known [12] that $\mu\mathcal{F}_\infty$ for $\mu > 1$ forms also a RPI set for the system (5.15) with disturbances in $\mathcal{W}_d$ and is even an ultimate bound if $A + BF$ is Schur. Selecting $\mu > 1$ large enough such that cl$\mathcal{B} \subseteq \mu\mathcal{F}_\infty$ gives an ultimate bound for (5.15) satisfying the conditions of theorem 1. Hence, in this way an ultimate bound for (5.12)-(5.13) is obtained on the control update times. The forward algorithm of [48] can be used to compute $\mathcal{F}_\infty$. If $\mathcal{W}_d$ contains the origin in its interior, then it is even known that the algorithm terminates in finite time (as $A + BF$ is Schur).

Besides the forward algorithm to find $\mathcal{F}_\infty$, there are various other ways to compute RPI sets for discrete-time linear systems, see e.g. [12, 13, 47, 48, 64]. We will present here one approach based on ellipsoidal sets as in [48]. To use the ellipsoidal approach of [48], we assume that $\mathcal{W}_d$ is included in the ellipsoid of the form $\mathcal{E}_{R^{-1}} := \{w \mid w^T R^{-1} w \leq 1\}$ with $R > 0$. Techniques to find such an over-approximation are given in [15].

Along the lines of [48] it can be shown that feasibility of

$$P - \gamma^{-1} A_{cl} P A_{cl}^T - (1-\gamma)^{-1} R > 0 \text{ and } P > 0 \tag{5.40}$$

for some $\gamma \in (0, 1)$ yields (using Schur complements) that

$$(A_{cl}x + w)^T P^{-1} (A_{cl}x + w) < \gamma x^T P^{-1} x + (1-\gamma) w^T R^{-1} w.$$

From this it is easily seen that $x^T P^{-1} x \leq 1$ and $w^T R^{-1} w \leq 1$ imply $(A_{cl}x + w)^T P^{-1} (A_{cl}x + w) \leq 1$. This shows that $\Omega = \{x \mid x^T P^{-1} x \leq 1\}$ is a RPI set for (5.15). By suitable scaling such that cl$\mathcal{B} \subseteq \mu\Omega$ for $\mu > 1$ again an ultimate bound is obtained for the event-driven system (5.12)-(5.13) on the control update times.

5.9.2 Computational aspects for the uniform case

Also for PWL systems several ways to compute invariant sets are available [51, 63].

For the higher-order bimodal PWL system (5.18), we observe that in the first mode the x-evolution is given by $x_{k+1}^d = (A+BF)x_k^d + w_k^d$. This means that when $x_{k_0}^d \notin \mathcal{B}$ for some k_0 the trajectory will eventually satisfy $x_{k+1}^d \in \mu\mathcal{F}_\infty$ for a $\mu > 1$ with cl$\mathcal{B} \subseteq \mu\mathcal{F}_\infty$. $\mathcal{F}_\infty$ is again the smallest RPI set containing 0 for $x_{k+1}^d = (A + BF)x_k^d + w_k^d$ and disturbances in $\mathcal{W}_d$. However, the set $\mathcal{F}_\infty$ can be replaced by any other RPI set for the linear system containing 0 (e.g. based on ellipsoidal sets as in the previous section). Hence, any state trajectory x^d for the bimodal PWL system (5.18) reaches

the set

$$\left\{ \begin{pmatrix} (A+BF)x_k^d + w_k^d \\ Fx_k^d \end{pmatrix} \mid (A+BF)x_k^d + w_k^d \in \mu\mathcal{F}_\infty,\ w_k^d \in \mathcal{W}_d \right\}.$$

at some point. If one constructs now a RPI set Γ for the system (5.18) containing the above set, then $\Pi_n(\Gamma)$ is an ultimate bound for the event-driven system (5.12) and (5.14) on the control check times.

In case of the lower-order PWL model we will present an approach based on ellipsoidal sets although techniques using reachability analysis can be exploited as well. Actually, the example in section 5.11.2 uses both the ellipsoidal and the reachability approach for illustration purposes.

Theorem 7 *Consider the event-driven system* (5.12) *and* (5.14) *without disturbances. Let* $P > 0$ *be a solution to* $A_{cl}^T P A_{cl} - \gamma P < 0$ *for some* $\gamma \in (0,1)$*. Take* α^* *small such that* $\alpha^* > \max_{1,\ldots,p_{\max}} \sup\{x^T P x \mid x \in h_p(D_p)\}$ *and* $\alpha^* > \max\{x^T P x \mid x \in cl\mathcal{B}\}$*, where* $h_p(D_p)$ *denotes the image of the map* h_p *with its arguments in* D_p*. Define the set* $\Omega(\alpha^*) := \{x \mid x^T P x \leq \alpha^*\}$*. Then the PWL system* (5.27) *and consequently the event-driven system* (5.12) *and* (5.14) *on the control check times are ultimately bounded to the set* $\Omega(\alpha^*)$*.*

For brevity we omit the proof.

Also one could use techniques based on Input-to-State-practical-Stability for piecewise affine systems [52] or on robust convergence [34, Ch. 8.5] to compute ultimate bounds for (5.18), although these approaches typically rely on the fact that the 'local' dynamics around the origin is stable, which is typically not the case in our setting. Also these approaches do not use the structure of the problem at hand. In this section, we exploited the particular structure of the constructed PWL systems.

5.10 Tuning of the controller

In this section we indicate how the ultimate bound Ω depends on $\mathcal{B}$ for (5.12), thereby facilitating the selection of desirable ultimate bounds by tuning $\mathcal{B}$. We will present here results for the non-uniform case and for the *unperturbed case* with uniform sampling.

5.10.1 Non-uniform sampling

The following result can be inferred from [12].

Theorem 8 *Consider the system* (5.12)-(5.13) *with $\mathcal{W}_d$ a closed, convex set containing* 0, *F given and $\mathcal{B}$ an open set containing the origin.*

- *If Ω is a RPI set for the discrete-time linear system* (5.15) *containing cl$\mathcal{B}$, then for any $\mu \geq 1$ $\mu\Omega$ is a RPI set for* (5.15) *containing μcl$\mathcal{B}$.*
- *If the discrete-time linear system* (5.15) *is UB to Ω containing cl$\mathcal{B}$, then for any $\mu \geq 1$* (5.15) *is UB to $\mu\Omega$ containing μcl$\mathcal{B}$*

This result shows that Ω scales "linearly" with $\mathcal{B}$ for scaling factors larger than one. Consider the minimal RPI set $\mathcal{F}_\infty$ containing $\{0\}$. For small $\mathcal{B}$ this gives the ultimate bound[3] for the event-driven system on the control update times as long as the chosen cl$\mathcal{B}$ lies inside $\mathcal{F}_\infty$. If $\mathcal{B}$ is taken larger and cl$\mathcal{B}$ is not contained in $\mathcal{F}_\infty$ anymore, the linear scaling effect as in theorem 8 occurs. This effect is nicely demonstrated in the first example below.

For the tuning of the controller one typically selects the state feedback with gain F for arriving at suitable transient behavior. Indeed, outside $\mathcal{B}$ the dynamics is given by the discrete-time linear system (5.15), which implies that the *convergence* towards the ultimate bound is determined by F. Selecting F such that $A + BF$ has desired eigenvalues, yields a desired speed of convergence. If an ultimate bound Ω with cl$\mathcal{B} \subseteq \Omega$ is computed for a pre-selected $\mathcal{B}$, one tunes the size of the stabilization error $\mu\Omega$ by scaling $\mu\mathcal{B}$. However, a fundamental limitation is given by $\mathcal{F}_\infty$ as this is the error bound caused by the disturbances when $\mathcal{B} = \{0\}$. One cannot go beyond this ultimate bound without changing F, although still the effect of the disturbance remains present. However, in the *unperturbed* case any scaling factor holds for any $\mu > 0$ (as $\mathcal{F}_\infty = \{0\}$).

5.10.2 Uniform sampling for the unperturbed case

We consider the unperturbed case here ($\mathcal{W}_c = \{0\}$).

Theorem 9 *Consider the system* (5.12) *and* (5.14) *with $\mathcal{W}_c = \{0\}$ and $\mathcal{B}$ is open and contains the origin. If the PWL system* (5.27) *corresponding to $\mathcal{B}$ is UB to the*

[3] Strictly speaking, an ultimate bound is the set $\mu\mathcal{F}_\infty$ for any small $\mu > 1$ as $\mathcal{F}_\infty$ is only approached asymptotically by some trajectories of the discrete-time linear system (5.15).

positively invariant set Ω and $\mathcal{B} \subseteq \Omega$, then for any $\mu > 0$ (5.27) *corresponding to $\mu\mathcal{B}$ is UB to the positively invariant set $\mu\Omega$ and $\mu\mathcal{B} \subseteq \mu\Omega$*

Proof In the proof we will indicate the dependence of f_{PWL}, h_p and D_p on the set $\mathcal{B}$ via superscripts, i.e. $f^{\mathcal{B}}_{\text{PWL}}$, $h^{\mathcal{B}}_p$ and $D^{\mathcal{B}}_p$, respectively. Let $\mu > 0$. The mappings $h^{\mathcal{B}}_p$ do not depend on $\mathcal{B}$, only on p, the number of discrete-time steps the control value is held. Hence, $h^{\mu\mathcal{B}}_p = h^{\mathcal{B}}_p$. This yields together with the linearity of the mappings that $D^{\mu\mathcal{B}}_p = \mu D^{\mathcal{B}}_p$ and $D^{\mu\mathcal{B}}_\infty = \mu D^{\mathcal{B}}_\infty$. Hence, $f^{\mu\mathcal{B}}_{\text{PWL}}(\mu x) = \mu f^{\mathcal{B}}_{\text{PWL}}(x)$. Indeed, if $x \in D^{\mathcal{B}}_p$, then $\mu x \in \mu D^{\mathcal{B}}_p = D^{\mu\mathcal{B}}_p$. As a consequence, it holds that $f^{\mu\mathcal{B}}_{\text{PWL}}(\mu x) = h^{\mu\mathcal{B}}_p(\mu x) = \mu h^{\mathcal{B}}_p(x) = \mu f^{\mathcal{B}}_{\text{PWL}}(x)$. The same reasoning can be applied to $x \in \mathcal{B}$ and $x \in D^{\mathcal{B}}_\infty$. If we denote the state trajectory $x^{d,x_0,\mathcal{B}}$ of the system (5.27) corresponding to $\mathcal{B}$ from initial state x_0, then we obtain the relation $x^{d,\mu x_0,\mu\mathcal{B}} = \mu x^{d,x_0,\mathcal{B}}$. From the latter relationship, the result in the theorem follows. □

This theorem gives a means, similarly to the non-uniform case, to tune the ultimate bound by suitably selecting the event-triggering mechanisms parameterized by $\mathcal{B}$. Scaling $\mathcal{B}$ with a constant $\mu > 0$ leads to an ultimate error bound that is μ times the bound belonging to $\mathcal{B}$. Note that due to the absence of perturbations, we can scale $\mathcal{B}$ with any $\mu > 0$ instead of $\mu > 1$.

Analogous results can be derived for the bimodal PWL system (5.18) without disturbances.

5.11 Examples

5.11.1 Non-uniform sampling

To illustrate the theory in case of *non-uniform* sampling (5.13) we will use the example (5.1) of section 5.2 with $F = -0.45$. Note that in the introduction we used *uniform* sampling. In figure 5.5 the ratio of the number of control updates in comparison to the case where the updates are performed each sample time (i.e. $u^d_k = -0.45x^d_k$ for all x^d_k) and the maximal value of the state variable (after transients) $x_{\max} := \limsup_{t\to\infty} |x(t)|$ (the "minimal ultimate bound"), respectively, versus the parameter e_T are displayed, where $\mathcal{B} = \{x \mid |x| < e_T\}$.

The figure of the ultimate bounds can nicely be derived from the theory. First, we

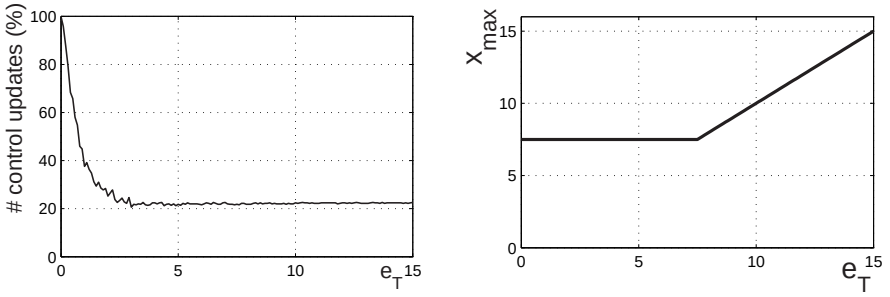


Figure 5.5: The control effort and $x_{\max}$ versus e_T for the example of section 5.2.1.

compute for the system (5.1), the discretized version (5.15) with sample time $T_s = 0.1$:

$$x_{k+1}^d = 1.051x_k^d + 1.025u_k^d + w_k^d;\ u_k^d = -0.45x_k^d \tag{5.41}$$

or

$$x_{k+1}^d = 0.590x_k^d + w_k^d \tag{5.42}$$

with $3.076 \leq w_k \leq 3.076$. The minimal RPI set $\mathcal{F}_\infty$ for (5.42) containing $\{0\}$ is equal to the "ellipsoid" $[-7.50, 7.50]$. Hence, note that as long as $e_T < 7.50$ the ultimate bound of the system (5.12)-(5.13) is equal to $\mathcal{F}_\infty$ (or strictly speaking to the set $\mu\mathcal{F}_\infty$ for a small $\mu > 1$ as discussed in the footnote in section 5.10). This explains the constant line in the $x_{\max}$ versus e_T plot in figure 5.5 up to $e_T = 7.50$. At the moment e_T becomes larger than 7.50, the condition of theorem 1 that cl$\mathcal{B} \subset \mathcal{F}_\infty$ does no longer hold. However, we can now use the "scaling effect" from theorem 8. Theorem 8 implies that $\frac{e_T}{7.50}\mathcal{F}_\infty$ is RPI and the linear system (5.42) is UB to $\frac{e_T}{7.50}\mathcal{F}_\infty$ when $e_T > 7.50$. Since cl$\mathcal{B} \subseteq \frac{e_T}{7.50}\mathcal{F}_\infty$ holds, theorem 1 implies that $\frac{e_T}{7.50}\mathcal{F}_\infty$ is RPI for (5.42) and the event-driven system (5.12)-(5.13) is UB to $\frac{e_T}{7.50}\mathcal{F}_\infty$. This explains the linear part in the $x_{\max}$ versus e_T plot in figure 5.5. Hence, we can reduce the number of control updates with almost 80% in this set-up without reducing the control accuracy (e.g. take $e_T = 5$)!

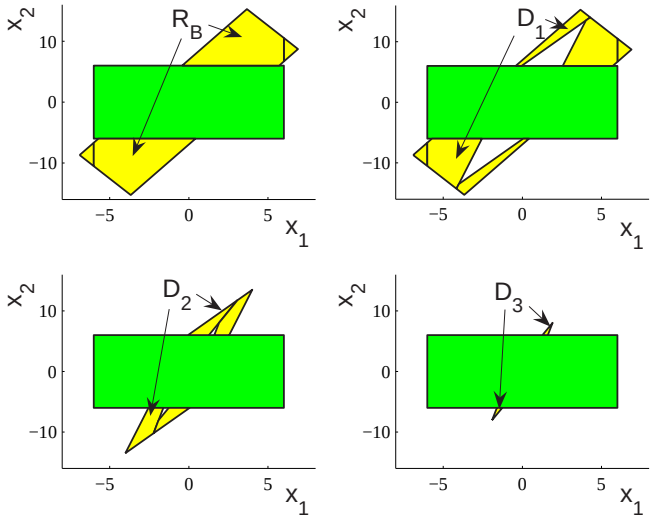


Figure 5.6: Sets $R_{\mathcal{B}}$, D_1, D_2, D_3 in light grey and set $\mathcal{B}$ the dark grey rectangle.

5.11.2 Uniform sampling

To demonstrate the results of section 5.7.2.1 for uniform sampling, we have taken the example of an unstable system with two states ($n = 2$) given by (5.12) with

$$A_c = \begin{bmatrix} 1070 & 270 \\ 270 & 40 \end{bmatrix} \quad B_c = \begin{bmatrix} 453 \\ 874 \end{bmatrix} \tag{5.43}$$

The controller matrix is taken to be $F = [-2.4604 \ \ -0.2340]$. The matrices in the discrete-time version (5.15) are equal to

$$A = \begin{bmatrix} 3.00 & 0.50 \\ 0.50 & 1.10 \end{bmatrix} \quad B = \begin{bmatrix} 1.00 \\ 1.00 \end{bmatrix} \tag{5.44}$$

for $T_s = 0.001$. Note that the the eigenvalues of $A_{cl} = A + BF$ are $0.7 \pm 0.7i$ and of A are 0.97 and 3.12. $\mathcal{B} = \{x \mid |x_1| < e_T, |x_2| < e_T\}$ with $e_T = 6$. We computed $p_{\max}$ by continuously increasing its value and we reached the equality $R_{\mathcal{B}} = \bigcup_{i=1}^{p_{\max}} D_i$ (which implies that $D_\infty = \emptyset$). We find $p_{\max}$ equal to 3. Figure 5.6 displays the calculated sets $R_{\mathcal{B}}$ and D_p, $p = 1, 2, 3$ as given by equation (5.23).

The dynamics that are valid inside D_p, calculated with equation (5.22) are:

$$\begin{aligned} h_0(x_l^d) &= \begin{bmatrix} 0.537 & 0.264 \\ -1.96 & 0.863 \end{bmatrix} x_l^d \\ h_1(x_l^d) &= \begin{bmatrix} -1.82 & 0.985 \\ -4.34 & 0.843 \end{bmatrix} x_l^d \\ h_2(x_l^d) &= \begin{bmatrix} -10.1 & 3.13 \\ -8.12 & 1.18 \end{bmatrix} x_l^d \\ h_3(x_l^d) &= \begin{bmatrix} -36.6 & 9.73 \\ -16.4 & 2.62 \end{bmatrix} x_l^d \end{aligned} \tag{5.45}$$

As could be expected, the dynamics corresponding to h_0 is asymptotically stable, while the dynamics corresponding to h_1, h_2 and h_3 are unstable. Note that typical approaches for stability analysis of PWL systems, based on common quadratic or piecewise quadratic Lyapunov functions, fail in this situation.

Since we have obtained the PWL-description of the system we can apply the theory presented in section 5.9.2. Using the ellipsoidal approach as presented in theorem 7 we obtain the ellipsoid Ω in figure 5.7. We also computed the reachable set Ω_{reach} for the PWL system from points in $R_{\mathcal{B}}$. For the computation of this set a combination of tools from [51] and [47] was used. Note that Ω_{reach} is a positively invariant set for the PWL system. Since $\mathcal{B} \subset \Omega_{\text{reach}}$ and outside $\mathcal{B}$ the dynamics on the event times is equal to $x_{k+1}^d = A_{cl} x_k^d$, similar statements can be made for Ω_{reach} as for Ω.

Figure 5.7 also shows a time simulation of the continuous time system. A dotted line shows the intersample behavior in which the small diamonds indicate the values at the control check times. It can be seen that the trajectory is not restricted to the depicted Ω_{reach} (in dark grey), due to the intersample behavior. Bounds on the intersample behavior can be obtained via section 5.8.

5.12 Conclusions

Although in many practical control problems it is natural and logical to use event-driven controllers, their application is scarce in both industry and academia. A major reason why time-driven control still dominates is the absence of a system theory for event-driven systems. However, due to the various benefits of event-driven control, it is worthwhile to overcome the difficulties in the analysis of this type of control. This chapter aims at using event-driven control to reduce the required (average) pro-

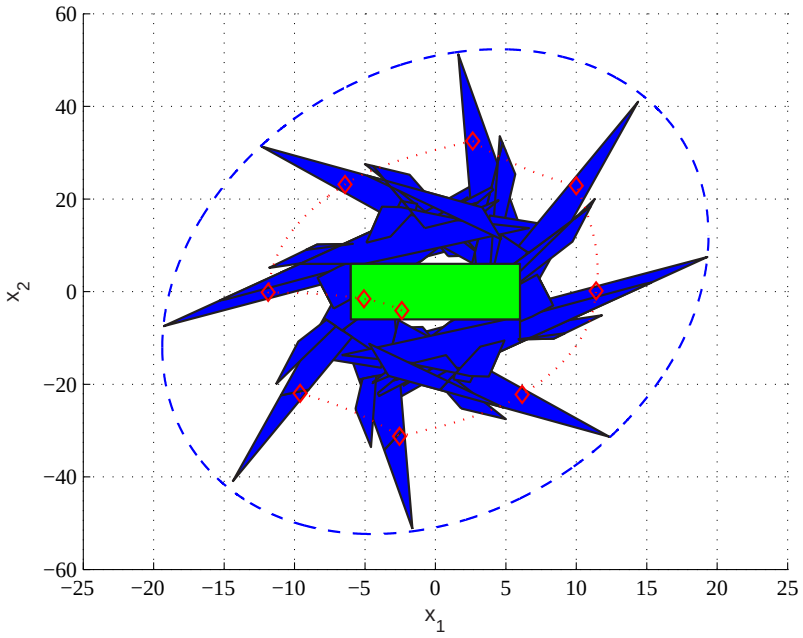


Figure 5.7: Ellipsoid Ω indicated by the dashed line, set $\Omega_{\min}$ in dark grey and set $\mathcal{B}$ in light grey.

cessor load for the implementation of digital controllers. An introductory example already illustrated the reduction of control computations (up to 80%) that is achievable. That this reduction of control computations indeed leads to a significantly lower processor load, in spite of the introduced overhead of the event-triggering mechanism, will be experimentally validated in the following chapter. However, one still has to make the trade-off between this reduction in resource utilization on one hand and the control performance on the other. This chapter provides theory that gives insight in the control performance for a particular event-driven scheme. The control performance is expressed in terms of ultimate bounds and speed of convergence to this bound. It is shown how these properties depend on the parameters of the control strategy. The results are based on inferring properties (like robust positive invariance, ultimate boundedness and convergence indices) for the event-driven controlled system from discrete-time linear systems (in case of non-uniform sampling) or piecewise lin-

ear systems (in case of uniform sampling). We presented computational means and tuning rules that support the design of these controllers.

Although this chapter analyzes a particular event-driven control structure, it already indicates the complexity and challenges for the analysis and synthesis of these type of control loops. Given the advantages of event-driven controllers and the various sources of event-triggering mechanisms present in industrial practice, it is fruitful to continue this line of research and developing a mature event-driven system theory. Future work will focus on the finite number of regions of the piecewise linear model, on tuning theory for the perturbed event-driven system with the uniform mechanism and on extending the current work to include reference tracking. From a broader perspective, we will consider also the analysis and synthesis of control schemes based on other event-triggering mechanisms like low resolution sensors as was initiated in [36].

6

Processor load for event-driven controllers[1]

6.1 Introduction

In most applications nowadays, digital controllers are implemented on embedded hardware with strong requirements for both the control performance as well as resource utilization like processor load. Often, a high update frequency of the control algorithm is chosen to be able to guarantee good control performance. This, however, evokes high processor loads. Conventionally, designers try to reduce the sample frequency of the digital controller as much as possible, to minimize the processor load, while keeping in mind the (minimal) required control performance. In almost all of the designs, the sample frequency is taken constant, creating a constant distribution of the processor load for the specific control task.

The sample frequency is normally chosen on the requirement to track fast changing reference signals or to reject high frequency disturbances. However, in many cases reference signals are not changing continuously and severe disturbances appear only sporadically. Only during these periods a high sample frequency is needed, while in other periods of time one does not have to require the same (high) sample frequency of the controller. This rationale indicates that it would be beneficial to vary the sample frequency to optimize over both the control performance and the processor load at the same time.

[1] This chapter is partially based on the work published in the proceedings of the IEEE Conference on Control and Applications 2006 [71] and is submitted for journal publication [74].

In literature [4, 22, 41, 66], event-driven (or asynchronous) controllers are proposed to make this trade-off between control performance and processor load. Specially designed event-generating mechanisms take care of triggering the controller to update the actuator signal at specific moments in time. Already in 1962, research was published in which a controller is presented with its sampling frequency varying relative to the derivative of the error signal. It was shown by simulations that over a given time interval, fewer samples were needed with the variable sampling frequency system than with a fixed-frequency sampling system while maintaining essentially the same response characteristics. In [4], a similar idea is presented based on varying sample frequency PID control to reduce the processor load of the implemented algorithm. Simulations on a double-tank process show that it is possible to significantly reduce the number of control updates with only a minor control performance degradation. Henriksson and co-workers [41] use for example optimal controllers to distribute processing power between three controllers running at varying sample frequencies. The authors of [66] present a sample period dependent controller that regulates the processor utilization to avoid overload. In chapter 5 various control structures are analyzed in which the sample frequency is chosen relative to the absolute value of the measured tracking error. When the error is small, fewer or even no computations are carried out and the actuator signal is held constant. This should reduce the processor load at those periods of time. With this controller a trade-off can be created between control performance and processor load.

In the above mentioned literature, all indications of processor load reduction are obtained by simulating or analyzing only the update frequency of the controller. Several assumptions are made to relate the simulated number of control updates to the processor load, but experimental evidence has not been presented in literature so far. One common assumption is that only relatively few overhead is needed to implement the event-generating mechanism of the event-driven controllers. Furthermore, the influence of for instance context switches and varying communication loads are assumed to be of minor influence on the time the control algorithm needs to execute. For these reasons, the event-driven controller could even reduce the number of control updates, while increasing the processor load.

The purpose of this chapter is to experimentally validate the promise of event-driven controllers to reduce processor load. In particular, we will implement the type of event-driven controller as proposed in chapter 5. Both a time-driven as well as an event-driven controller are implemented on an experimental setup of a printer paper

path, driven by multiple DC-motors. Furthermore, we investigate the possibility to predict the processor load a priori, without having to implement the controller on a test setup. This will be done by investigating the relation between the reduced number of control updates and the resulting processor load. Hence, the trade-off between control performance and processor load can be made model-based.

This chapter is outlined as follows: Section 6.2 presents the event-driven control algorithm, that is used for simulations and experiments. Then, section 6.3 describes the experimental setup. In section 6.4 results of simulations are described. Based on these results, section 6.5 presents the prediction of the processor load. This prediction is compared with the measurement results, presented in section 6.6. Finally, discussion and conclusions are presented.

6.2 Event-driven controller

We consider a single input single output plant described by

$$\begin{aligned} \dot{x}(t) &= f(x(t), u(t)) \\ y(t) &= h(x(t)) \end{aligned} \tag{6.1}$$

where $x(t) \in \mathbb{R}^n$ is the state, $u(t) \in \mathbb{R}$ the control input and $y(t) \in \mathbb{R}$ the output, respectively, at time $t \in \mathbb{R}_+$. $f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}^n$ and $h : \mathbb{R}^n \to \mathbb{R}$ can be linear as well as non-linear functions.

To control the plant (6.1), such that good tracking behavior is obtained, we use a digital PI feedback controller, given by the following difference equation:

$$u_k = Pe_k + (IT_s - P)e_{k-1} + u_{k-1}, \tag{6.2}$$

where $e_k := y(kT_s) - r_k$ is the tracking error at time $t = kT_s$ and $r_k = r(kT_s) \in \mathbb{R}$ is the value of the reference signal r at time kT_s, $k = 0, 1, 2, ...$, T_s is the sample time. By using zero-order hold, $u(t) = u_k$ for all $t \in [kT_s, (k+1)T_s)$. P and I are the proportional and integral gain of the PI controller. This is the conventional digital setup, using a fixed sample time T_s is used, meaning that the *control update times* are equal to kT_s, $k = 0, 1, 2, ...$ We call this a *time-driven* controller.

To reduce the number of required control calculations and actuator signal updates, we propose that the control value is not updated if the error e_k at $t = kT_s$ is smaller than a threshold value e_T. If the error is larger than e_T, an update is performed according to (6.2).

Hence, the controller (6.2) is modified to

$$u_k = \begin{cases} Pe_k + (IT_s - P)e_{k-1} + u_{k-1} & \text{if } |e_k| > e_T \\ u_{k-1} & \text{if } |e_k| \leq e_T \end{cases} \tag{6.3}$$

where again $u(t) = u_k$ for all $t \in [kT_s, (k+1)T_s)$. This is the particular *event-driven* controller studied in this chapter. Whether the error is smaller than e_T, is still detected at a constant frequency (at times kT_s, $k = 0, 1, 2, ...$). These times we call the *control check times*. The control update times are here the times kT_s for which $|e_k| > e_T$. This is a *uniform sampled* event-driven controller as analyzed in chapter 5. Note that the value of T_s is not changed when for some samples no control update is performed.

Loosely speaking, the aim of the control design (selecting T_s, e_T, P and I) is to get good control performance (in the sense that the maximal tracking error $e_{max} := \max_{t \in \mathbb{R}_+} |y(t) - r(t)|$ is acceptable) and the processor load is small.

6.3 Experimental setup

6.3.1 Plant

Figure 6.1 depicts a photograph of the experimental setup. The setup was produced at the University of Twente for conducting experiments in a parallel project. It represents a part of the paper path of a printer that consists of 4 identical motors which drive 4 rollers. These rollers drive the sheets of paper through the paper path. The goal of

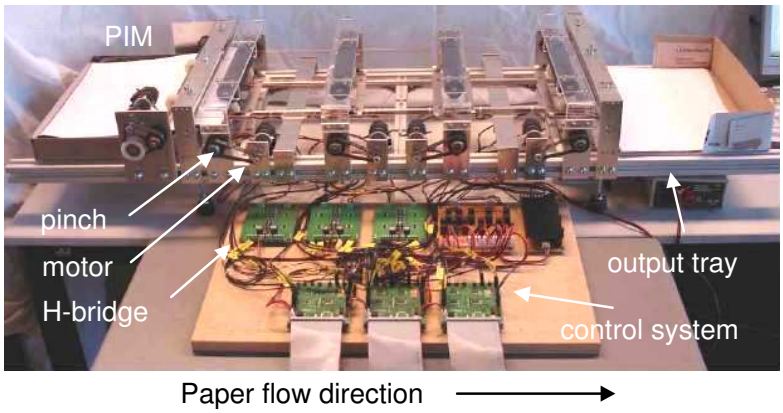


Figure 6.1: Photograph of the paper path setup.

the paper path is to transport sheets of paper from the paper input module (PIM), to the output tray. During a print job, all motors in the paper path accelerate to a certain constant velocity. Then, sheets of paper are injected into the paper path. At a certain location in the paper path, an image is fused onto the paper. Next, the paper is turned for duplex printing or the paper is transported to the output tray.

The particular motor used is the Maxon RE25 20 watt motor. The motor axes are coupled to the rollers with stiff belts. To the other end of each motor axis, a 500-slit rotary encoder is connected. This signal is acquired with quadrature demodulation, resulting in a resolution of 2000 counts per rotation. H-bridge amplifiers are used to control the motors. They operate at 22 volt and are limited to a maximum current of 3 ampère.

6.3.2 Control system

The digital control system consists of a PC104+ CPU (processor) board with a 600 MHz x86 compatible CPU, supplied with 256 MB RAM and a 32 MB Flash disk which contains the real-time (RTAI) operating system. An FPGA is connected to the CPU board via the PCI communication bus in order to perform the I/O operations. In this setup the configuration for the FPGA contains four pulse width modulated (PWM) outputs and four encoder quadrature inputs. The PWM output signals, that drive the H-bridge amplifiers, operate at a frequency of 16kHz. The duty-cycle of each PWM signal can be set in 2048 steps. Separate signal outputs determine the rotation directions of the motors. The CPU sets the duty-cycles and direction signals and the FPGA keeps these values until a new value has been received (implementing $u_k = u_{k-1}$ as in equation (6.3)). Each encoder input increments a separate counter at the FPGA on every edge received from the encoder. The CPU can read this counter.

6.3.3 Model

To simulate the behavior of one separate controlled motor, we created a model in the simulation package 20-sim (University of Twente, The Netherlands [88]). This model consists of an accurate description of the motor (delivered by the Maxon Motor company), a description of the load together with a non-linear friction model of the bearings, the PI feedback controller and the quantization effects caused by the encoder and H-bridge amplifier. In terms of equation (6.1) $y(t)$ is the angular velocity of the motor and $u(t)$ is the motor voltage.

6.3.4 Controller design

As the paper path consists of identical motors with the same functionality, we only explain the controller design for a single motor. At the setup, four separate but identical controllers are implemented to control the four motors.

To control the angular velocity of the motor, the time-driven PI controller as given in (6.2) was tuned to get good tracking behavior, using common design rules [25] and implemented on the test setup with $P = 0.1$ and $I = 1$. The resulting closed-loop response (from reference velocity r to output velocity y), depicted in figure 6.2, was derived from the frequency response of the sensitivity function. The sensitivity function was experimentally determined at the setup (using Welch's method [35]), by injecting white noise at the actuator signal and measuring the control output signal u. Note that the values for frequencies below 6 Hz should be considered uncertain, as the coherence of the sensitivity measurement was far below 1 for these frequencies (see [35] for details). In the figure, the bandwidth is indicated at approximately 20 Hz. Rules of thumb advise to set the controller frequency at a minimum of 6 times the closed-loop bandwidth, see e.g. [25, Ch. 11]. We chose the sample frequency at 100 Hz, which is only 5 times the obtained closed-loop bandwidth. The choice for this

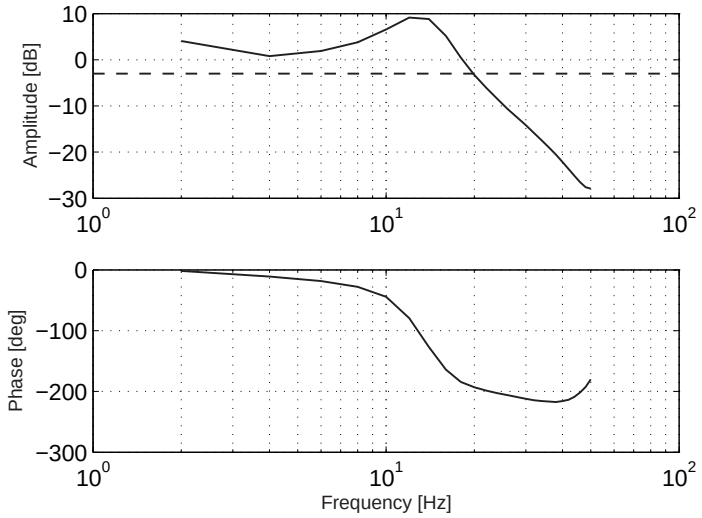


Figure 6.2: Closed-loop response of the motor.

low sample frequency was made to assure an initial low processor load for both the time-driven and the event-driven controller.

The event-driven controller, as given in (6.3), was implemented using the same sample frequency of 100 Hz. Furthermore, the controller parameters for the event-driven controller were chosen identical to the obtained parameters for the time-driven controller. This means that no additional tuning was performed.

The event-driven controller can be written in pseudo code as follows:

```
1  pos = input(encoder);
2  vel = (pos - previous(pos))/Ts;
3  error = reference - vel;
4
5  if (error > eT OR error < -eT) then
6    uP = P*error;
7    uI = previous(uI)+I*previous(error)*Ts;
8
9    u = limit(uP+uI, min_u, max_u);
10
11   motor_voltage = output(u);
12 end;
```

where "pos" is the measured position and "vel" is the derived velocity. In line 9 the computed controller output is limited to a 100% duty-cycle of the PWM signal, giving the maximum and minimum voltage of 22 and -22 volt respectively.

The time-driven controller was implemented in a similar way, by omitting the lines with numbers 5 and 12. Note that this indicates clearly the overhead introduced by the event-driven controller. The potential benefit for the event-driven controller can also be observed, as lines 6 to 11 are only executed under specific conditions.

6.4 Simulation results

Simulations are performed for one motor only, as the motors are of the same type. The simulation results for the time-driven controller are depicted in figure 6.3. The first graph gives the velocity reference signal in rotations per second (rps). In the setup, this profile can be used for every motor (but shifted over time) to drive several sheets of paper through the paper path sequentially (see section 6.6.3). The profile

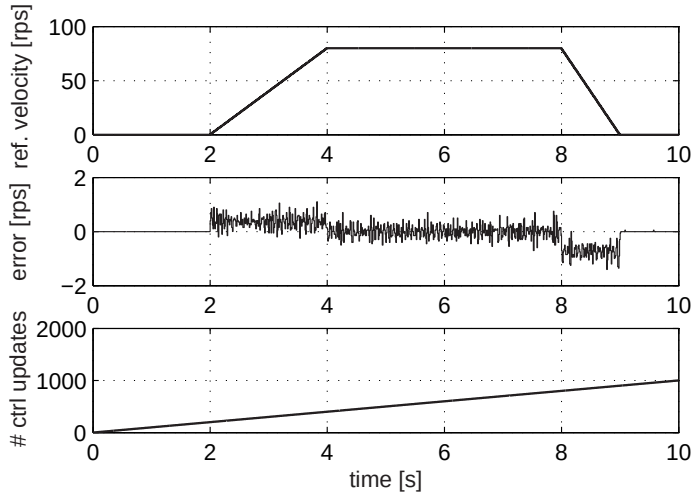


Figure 6.3: Simulation results of the time-driven controller.

starts and ends with a period of zero velocity. During this period, no sheets need to be transported and therefore the motor can halt. The second graph shows the error of the controller. When the motor is running at non-zero velocity, the error demonstrates a noisy behavior. This is caused by the belt that inserts relatively high frequent disturbances in the system. The third graph shows the number of control updates for the time-driven controller, which is linearly increasing in time with 100 updates per second for the 100 Hz controller.

The same signals are plotted in figure 6.4 for the event-driven controller simulation with $e_T = 0.9$ rps. The reference velocity is chosen the same, as for the time-driven controller. As expected, the second plot shows a larger error (up to 1.85 rps) compared to the time-driven controller simulation (error up to 1.2 rps). When the motor is running at constant velocity, the error stays below the specified bound e_T. The third graph shows how the number of control updates increases over the simulation. It can clearly be seen that when the motor is in stand-still, or when the motor is running at constant velocity, no control updates are needed. However, when more severe disturbances would be present, updates might also be necessary during the constant velocity phases. An example of such a disturbance could be a sheet of paper that is traveling through the paper path, creating a disturbance torque onto the motor. This

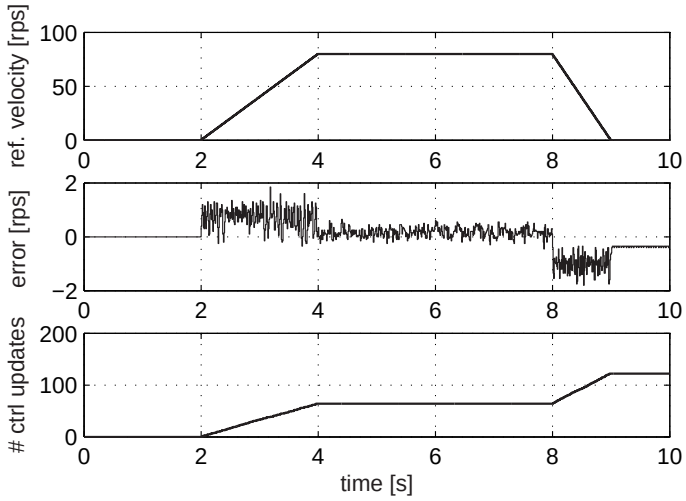


Figure 6.4: Simulation results of the event-driven controller for $e_T = 0.9$ rps.

situation will be studied later in this chapter. In the third graph it can also be seen that the controller takes more control updates per second when decelerating compared to accelerating. This can be explained by the fact that the controller is followed by a zero-order hold and that the motor decelerates faster than it is accelerating. To follow the reference velocity during deceleration with comparable performance, more actuator signal updates are needed per time interval.

As the control performance measure we use the maximum error (e_{max}) over each 10 seconds simulation, as described at the end of section 6.2. Naturally, when e_T is increased, the maximum error increases. This relation between e_T and e_{max} is investigated and depicted in the first plot of figure 6.5 (solid line). In this plot, the simulation results for 300 different values of e_T in the range [0, 2] are given. The straight dashed line in this figure visualizes e_{max} for the time-driven controller simulation (being 1.2 rps). This value is somewhat lower than the lowest maximum error that the event-driven controller can achieve, which is equal to 1.3 rps and realized for $e_T < 0.25$ rps. Note that when choosing $e_T = 0$ rps, the event-driven controller still differs from the time-driven controller. Indeed, when the error is *exactly* 0, the time-driven controller will implement the control value based on the error of 0 rps. The event-driven controller, on the other hand, will hold the last implemented control value, that was

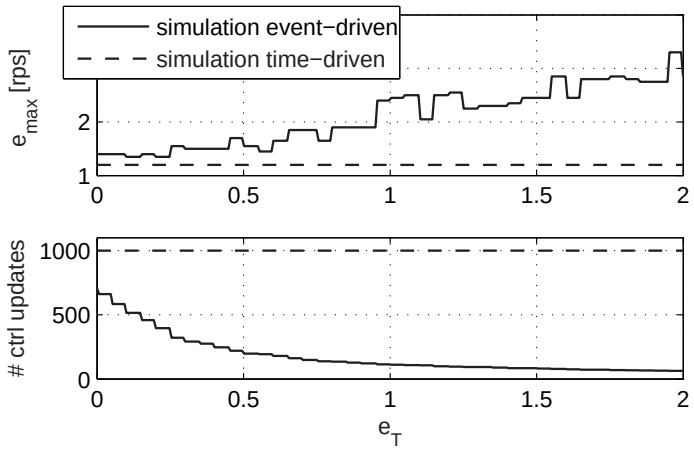


Figure 6.5: Simulation results of time-driven and event-driven controller. The event-driven controller was simulated for 300 values of e_T.

possible based on an error unequal to 0 rps.

It can be seen that the simulation results (e_{max}) are quantized in steps of 0.05 rps, because the position is measured at 2000 counts/rot at a frequency of 100 samples/s.

The solid line in the lower graph of figure 6.5 shows for each chosen e_T the total number of control updates needed in the simulation. When for the event-driven controller e_T is set to 0 rps, the total number of control updates already decreases from 1,000 to 700. This is because the reference velocity is zero for 3 seconds. No motor voltage needs to be applied to keep the motor in stand-still. This is only true when no excessive disturbance is present that could force the angular velocity of the motor to a non-zero value. The quantization effect can again be observed. As the error is quantized in steps of 0.05 rps, the total number of control updates only changes at values of e_T that are a multiple of 0.05 rps.

6.5 Prediction

From the number of control updates, obtained from the simulations, we can predict the processor load. The processor load is defined here as the processing time needed for a particular time-interval (in our case the 10 seconds time-interval). For this, we use the processing time needed to execute the particular tasks in the control algorithm. The

Table 6.1

multiplication of two floats	*	20.0 ns
division of two floats	/	140 ns
addition of two floats	+	16.7 ns
subtraction of two floats	-	16.7 ns
comparison of two floats	$<$	96.0 ns
IO action to update PWM duty cycle	IO	2.10 μs

Micro measurement numbers for PC104.

main tasks that can be distinguished in the event-driven control algorithm are: *input (lines 1-3)*, *check (lines 5 and 12)*, *calculation (lines 6-9)* and *output (line 11)*. The numbers above coincide with the line numbers of the pseudo-code in section 6.3.4. For the time-driven controller, *check* is omitted, but the other tasks are identical. The computation times associated with these tasks, are indicated by t_{input}, t_{check}, t_{calc} and t_{output}, respectively. The total processing time of the time-driven controller for a 10 seconds experiment (t_{td}^{10}), can be computed as follows:

$$t_{td}^{10} = 10 f_s (t_{input} + t_{calc} + t_{output}) \tag{6.4}$$

with f_s the sample frequency of the controller.

For the event-driven controller, the total processing time of a 10 seconds experiment (t_{ed}^{10}) is given as:

$$t_{ed}^{10} = 10 f_s (t_{input} + t_{check}) + c^{10} (t_{calc} + t_{output}) \tag{6.5}$$

with c^{10} the number of control updates over the 10 seconds experiment.

Quantitative estimates of the computation times for the individual tasks can be obtained from micro measurements, also called benchmark numbers [69]. These measurements give the time duration of individual basic operations, like e.g. the addition of two floating point numbers (floats), and depend mainly on the speed of the processor, memory and on the floating point unit. The tasks that run on the processor can be split up in terms of those basic operations. From this we can obtain the expected computation times of the tasks. Table 6.1 gives the micro measurement numbers for the PC104 board in the test setup. These are only the most time consuming numbers that are used for the presented controller. Operations like a binary OR are assumed to be executed in negligible time.

Table 6.2	
t_{input}	2.27 μs
t_{check}	192 ns
t_{calc}	389 ns
t_{output}	2.10 μs

Estimated computation times of the individual tasks.

Next, we analyzed the C-code that was synthesized from the controller model to obtain the set of basic operations from which the tasks are composed. For example, *input* consists of 1 IO action, 2 subtractions and 1 division. From these sets, in combination with the micro measurements numbers, we calculated the time that each task needs to execute. The results are given in table 6.2. The relatively large amount of time consumed by an IO operation is caused by context-switch time and the time it takes to communicate with the slower PCI-bus. In order to perform an IO operation, communication via a device-driver with the FPGA is necessary. A context-switch is made to and from the kernel-space to access the device-driver. t_{input} is estimated somewhat larger than t_{output}, because some additional processing is involved to derive the velocity measurement from the position data.

The computation times for the various operations can be assumed to be fairly constant. For this specific example, the controller is the only real-time task running and its size allows it to run entirely from cache memory, so no variations are expected.

Using equations (6.4) and (6.5), combined with the estimated computation times of the individual tasks, we are able to predict the total computation times of both the time-driven and the event-driven control algorithm for various values of e_T. From the simulation results, depicted in the bottom graph of figure 6.5, we obtained the number of control updates as the value for c^{10} in (6.5). The results are given in figure 6.6.

From equations (6.4) and (6.5) we can also derive the maximal achievable gain in processor load for the event-driven controller, compared to the time-driven controller. The maximal gain is obtained when we take $c^{10} = 0$. This implies that no control updates are needed to keep the error within the bounds of e_T. The maximally achievable

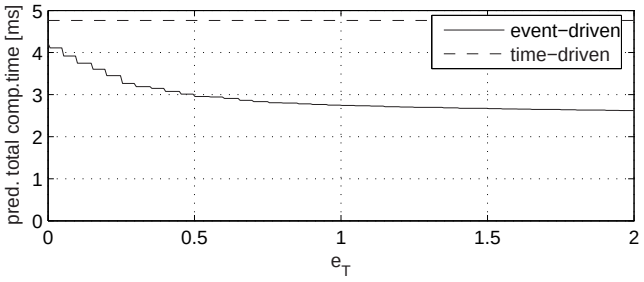


Figure 6.6: Predicted computation time for time-driven and event-driven controller using various values of e_T for the event-driven controller.

gain G_{max} in this particular setup is:

$$\begin{aligned} G_{max} &= \frac{10 f_s (t_{input} + t_{calc} + t_{output})}{10 f_s (t_{input} + t_{check}) + c^{10}(t_{calc} + t_{output})} \\ &= \frac{t_{input} + t_{calc} + t_{output}}{t_{input} + t_{check}} \approx 2 \end{aligned} \tag{6.6}$$

This can be verified in figure 6.6, as the time-driven controller uses approximately 5 ms and the event-driven controller 2.5 ms for large values of e_T.

6.6 Experiments

6.6.1 Processor load measurement

To measure the processor load of the control algorithm, we measure the time the processor needs from the start of *input*, to the end of *output*. For this purpose, we take

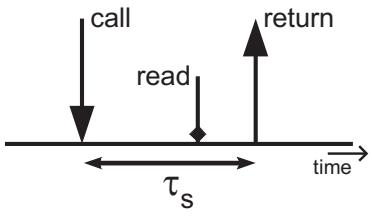


Figure 6.7: Illustration of the process to take a time-stamp.

two time-stamps; the first before line 1 in the pseudo-code, and the second after line 12. By subtracting the first time-stamp from the second, we obtain the elapsed time.

In order to take a time-stamp, a function is called which reads a time counter and returns its value (see figure 6.7). As indicated in this figure, time will elapse between the *call* and the *read* and between the *read* and the *return*, as on an x86-compatible CPU, time measurement is not atomic. The total time it takes to take a time-stamp, which is not necessarily constant over time, is called τ_s.

To perform correct time measurements, we need to subtract τ_s from each obtained time measurement. To obtain τ_s we take two successive time-stamps. We do this at every sample (after the second time-stamp has been taken), to obtain a recent value of τ_s and to check the variance over time. The time difference of the two values returned is equal to τ_s. To guarantee that the action of taking a time-stamp is not interrupted, it is assigned to the highest priority. For the particular setup τ_s was measured to be 0.98 μs with a maximum variation of $\pm 3\%$.

6.6.2 Experimental results for one motor

The measurement results of the experiment with one motor are depicted in figure 6.8. Here we chose $e_T = 0.9$ rps. One can compare these plots with the simulation results depicted in figure 6.4, as the same value for e_T was chosen and the same reference velocity was used. Note that no sheet disturbances are active yet. The second plot of the figure, which depicts the velocity error, shows that the results for the experiment are similar to the simulation results. This is also the case for the third plot, in which the number of control updates are plotted. The fourth plot shows the measured execution time of the control algorithm at each sample time, i.e. every 0.01 second. The offset, estimated at 2.5 μs (which is $t_{input} + t_{check}$), can clearly be distinguished. The extra time at the moments of the peeks in the plot during non-zero velocity in the reference signal, is the time that is needed for *calc* and *output* (estimated at: $t_{input} + t_{check} + t_{calc} + t_{output} = 4.9\mu s$).

The performance results (e_{max}) of 20 experiments for different values of e_T are depicted in the first graph of figure 6.9, together with the simulation results. The experimental results are similar to the simulation results as we observe the same lower bound and trend. The maximum error of the time-driven controller experiment was also the same as obtained from simulation. The second graph of figure 6.9 shows the number of control updates for these 20 experiments, together with the simulation results. Again, a nice fit between simulation and experiment is observed.

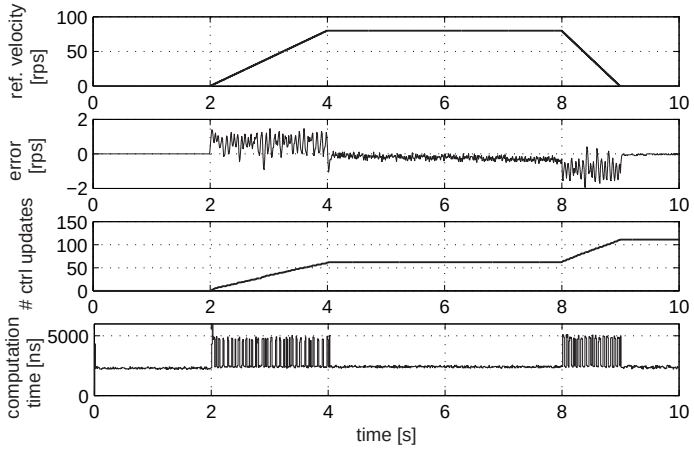


Figure 6.8: Experimental results of the event-driven controller for $e_T = 0.9$ rps.

The cumulated measured computation time for the same 20 experiments is depicted in figure 6.10, together with the predicted computation times. This is also shown for both the time-driven and the event-driven controller. It can be observed that the measurement results match the predictions closely.

6.6.3 Experimental results for paper path

To show the real industrial advantage of the event-driven controller, we have implemented the controller for every motor in the paper path, and inserted sheets in the path. The reference velocities are depicted in figure 6.11. The ramps of the profiles are chosen such that they do not coincide. This choice for the profiles is common in industry from a power perspective, as during the ramp-up phases the motors consume the most power. By separation of these ramps the peak power is kept low. For the presented event-driven controller, this also has a benefit for processor load, because the most processor power is needed during acceleration and deceleration.

When all motors have reached their steady state velocity, sheets are ejected from the Paper Input Module (PIM, see figure 6.1) into the first roller of the paper path at times 3.00, 3.25, 3.50, 3.75 and 4.00 s. The effect can clearly be observed in figure 6.12 which shows the error signals of the four motor controllers in the paper path (note the different scale of the magnitude in the first plot). Especially the first motor, of

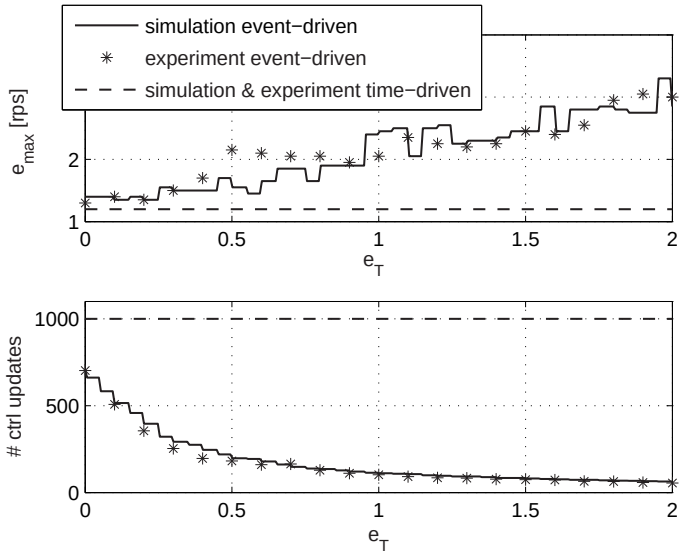


Figure 6.9: Experimental and simulation results of time-driven and event-driven controller. The event-driven controller was measured for various values of e_T.

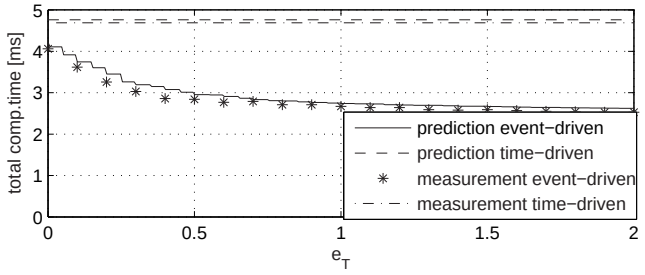


Figure 6.10: Experimental and predicted results of computation time for time-driven and event-driven controller using various values of e_T.

---

which the error signal is depicted in the first graph, shows peaks in the error as the speed of the PIM is not synchronized with the speed of the motors in the paper path. The first motor corrects this problem and therefore the error signal of the other motors show much less reaction to incoming sheets. In the first two graphs of figure 6.12 it can be seen that after the motor has accelerated, the error stays smaller than e_T for most of the time until the sheets are inserted. The tracking errors of motor 3 and 4

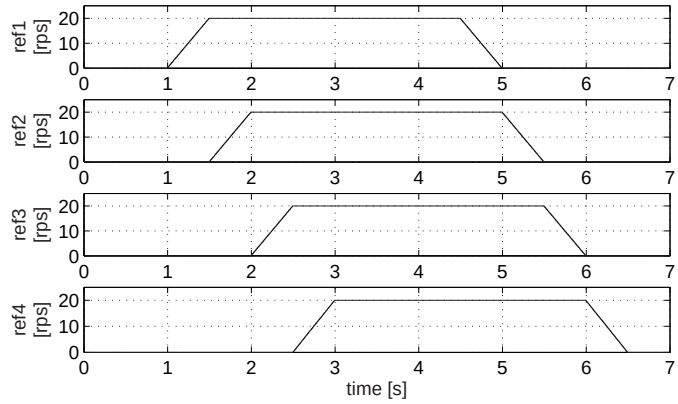


Figure 6.11: Reference signals for the 4 motors in the paper path.

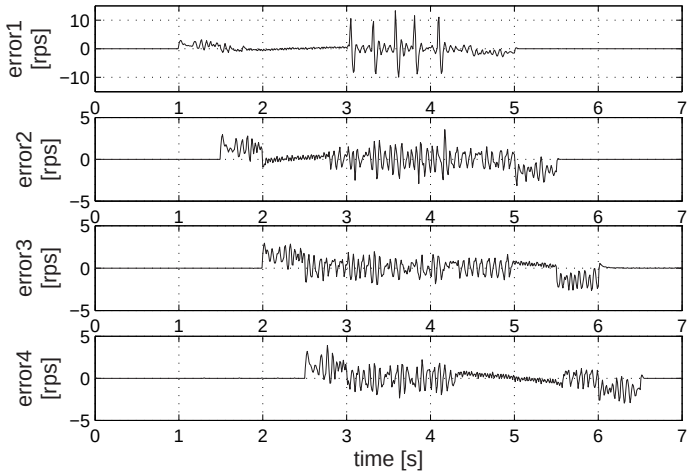


Figure 6.12: Error signals for the four motors in the paper path.

are only smaller than e_T in the period after the sheets have been handled but before the motors start decelerating. The obtained errors are acceptable to transport sheets of paper through a printer.

In figure 6.13 we show the total computation time of all four controllers for each

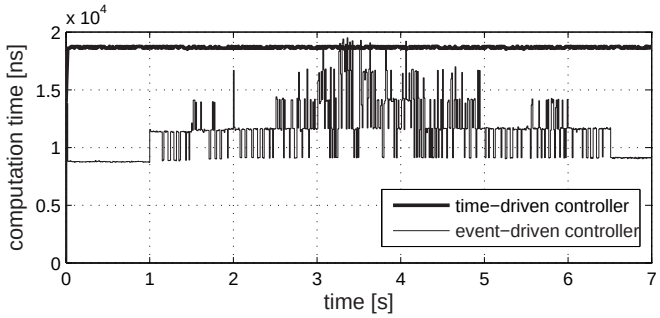


Figure 6.13: Added computation time for all motor controllers per sample.

sample moment. Also the computation time for the implementation with four time-driven controllers is depicted, which shows a fairly constant load of approximately 18 μs every sample (at 100 Hz). For the event-driven controller case an offset can be distinguished, like in the bottom graph of figure 6.8. Because at every sample moment four encoder counter values need to be communicated, the offset is here four times higher: 8.8 μs. For each controller that is active in a specific sample moment, approximately 2.6 μs is added to the computation time. It can be seen that indeed during the acceleration and deceleration phases, mostly just one of the four controllers is active. When sheets travel through the paper path, we see that all controllers need to be active to compensate for the disturbances, but still at a lower pace than the time-driven controller. For short periods of time, this results in the situation where all four controllers are active and therefore a computation time of 19.5 μs, which is slightly higher than for the time-driven controller. If we add all the computation times for every sample over the whole experiment time of 7 seconds, we obtain that it took 7.88 ms in total to execute the controllers (versus 12.6 ms for the time-driven controller).

When looking more carefully at the results, we see that on the average every controller only uses at maximum 50% of all control check times to perform an update, also at peak loads. This leads to the conclusion that we could schedule the controllers in a smart way such that we also reduce the peak load of the total computation time. The major advantage of this result is that we could possibly choose for a smaller processing unit or to schedule more tasks on the same processor. For this reason we scheduled the four controllers such that controller 3 was only executed if controller 1 was not executed in the same sample. In the same way, controller 4 was only executed when

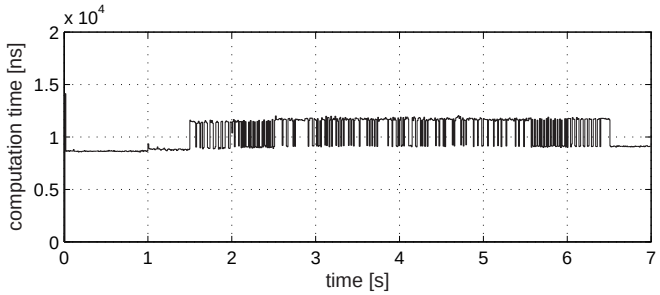


Figure 6.14: Added computation time for all motor controllers per sample for the scheduled case.

controller 2 was not executed in the same sample. With this implementation, at maximum two controllers can be executed during the same sample, resulting in a lower peak load of the processor. Moreover, when controllers 3 or 4 are not executed, the *input* and *check* for these controllers do not have to be executed as well. This means that at those moments the offset of 8.8 μs will be lowered too with 2.2 μs per controller (resulting in 6.6 μs or 4.4 μs respectively).

The resulting computation time per sample for this implementation is depicted in figure 6.14. It can be seen that the peak load is reduced from 19.5 μs to 12 μs. For the whole experiment we obtained a total computation time of 7.20 ms. Note that the proposed scheduling of the controllers involves that controllers 3 and 4 are sometimes delayed. This, however, did not have significant influence on the performance of the system, as the introduced delay for controllers 3 and 4 was not more than one or, occasionally, 2 samples.

6.7 Discussion

When comparing the results of the time-driven controller with the results of the event-driven controller, we observe a reduction up to 95% in the number of control updates, for high values of e_T ($e_T = 2$ rps). This resulted in a saving of the processor load of only 46%, due to the relative high offset caused by the value of t_{input} (as this has to be performed at 100 Hz in the event-driven controller as well). When we choose for example $e_T = 0.4$ rps in the presented application, we already obtain a saving in the processor load of 39%. This only increased the maximum error from 1.2 rps to 1.7 rps.

Table 6.3

	PC104	AVR	Pentium	DSP
*	20.0 ns	21.7 μs	2.46 ns	1.04 μs
/	140 ns	66.3 μs	16.0 ns	7.03 μs
+	16.7 ns	18.4 μs	1.55 ns	1.33 μs
-	16.7 ns	19.8 μs	1.75 ns	1.39 μs
$<$	50.0 ns	13.1 μs	8 ns	0.879 μs
IO	2.10 μs	0.214 μs	2.00 μs	0.392 μs

Micro measurement numbers for several processing platforms.

Of course, these figures depend heavily on the chosen setup. Important aspects of the setup in this context are: the complexity of the control algorithm, the processing platform together with the communication mechanisms, the reference signal to be tracked and the disturbances acting on the plant. For the experiments we used a simple (PI) control algorithm, that does not need much processing power to execute (i.e. t_{calc} is small). If we choose a more complex control algorithm, the savings in processor load increase. On the other hand, the *check* was also chosen simple (t_{check} small). A more complex check would have increased the overhead for the event-driven controller, and the savings in processor load would decrease. For the presented application, most processing time was assigned to the input and output actions. The largest processor load reduction was therefore caused by the reduced number of actuator updates that had to be performed. The reduction was bounded due to the constant number of input actions that had to be executed.

To investigate the dependence of the processing platform on the processor load, we performed the same micro measurements on several other platforms. The results are given in table 6.3. The first column repeats the data for the PC104 as used in the case study (see also table 6.1). For the data in the second column we used an 8-bit RISC micro controller from Atmel: AVR ATmega32, running at a clock frequency of 8 MHz. The third benchmark was performed at a high speed office PC: Pentium 4 processor, 3.40 GHz, hyper-threading CPU. The last benchmark was performed on a DSP: ADSP-21992, which is a mixed signal DSP controller suitable for a variety of high performance industrial motor control and signal processing applications, running at a clock frequency of 16 MHz. For the DSP and the AVR, no operating system was used because of the limited memory capacity.

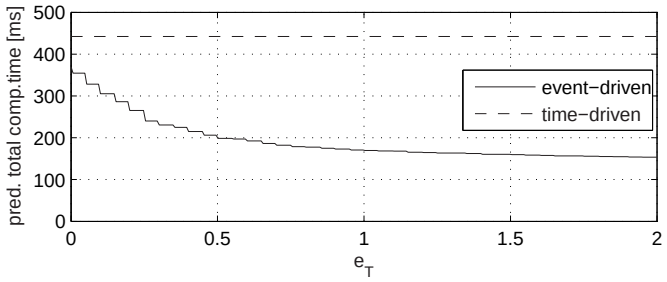


Figure 6.15: Predicted computation time for time-driven and event-driven controller running on the AVR micro controller.

Figure 6.15 gives the prediction of computation time for one single motor in which the controller would run on the 8-bit AVR micro controller. It can be seen that in this situation a maximum saving in processor load of 65% can be achieved for the event-driven controller, with $e_T = 2$ rps. This saving is still limited compared to the reduction in the number of control updates due to the time needed to perform the division of two floating point numbers in the *check*. For the Pentium, we could save at maximum 53% and for the DSP 50% for the considered situation.

The considered examples are just four typical examples out of many possible alternatives. It can be seen however that the gain in processor load can be easily predicted for each new situation. If sensor and actuator data have to be communicated over a network with limited bandwidth, savings of these kinds might be considered as well. As communication busses have limited bandwidth, reducing the buss load is beneficial for the total system. Moreover, when wireless communication is used, lower buss loads also save energy. Especially for battery-powered devices, this is an important aspect as wireless communications is a severe power consumer. Lots of research is carried out in the reduction of power usage for battery-powered devices like wireless sensors [23, 82].

When applying the considered event-driven controller for a single motor, we only decrease the average processor load and not the peak load. Therefore, it should be noted that the processing power that comes available temporarily should also be used to create an advantage for the total system. An example to reduce the overall processor load, is the case in which soft-realtime tasks (e.g. image processing), running on the same processor, can use the released processing power. It is the task of the scheduler

on the real-time operating system to take care of this.

In the example in which we control all 4 motors of the paper path setup, we see that the possible gain is also related to the moments at which reference signals change and disturbance is present. For the presented situation it is clear that the controller does not need to 'work' equally hard at every moment in time. In applications where the controller needs to run at a certain sample frequency continuously to keep the error within the required bounds, this specific event-driven control algorithm will not show the same benefit as in the presented application. The presented example however, is one that is representative for many controlled industrial systems found in practice.

6.8 Conclusions

The contribution of this chapter is twofold:

1. The potential of event-driven controllers was validated on an industrial setup.
2. The relation between reduced number of control computations and a lower processor load was studied.

This chapter *experimentally* validated the potential of event-driven controllers. Experiments showed that event-driven controllers can be used in practice to reduce the processor load by a factor of almost 2, when compared with conventional time-driven controllers. This involved only a small degradation of the control performance. We also argued that for the particular controller setup on a different processing platform, the processor load could have been reduced by a factor of 3, which shows the value of event-driven controllers and future research in this domain.

From simulation results, we were able to predict the processor load in the experiment accurately. This was done with relative low effort, despite the fact that many complex implementation factors are to be accounted for. For this purpose, micro measurements were used to estimate the processor load of the various tasks of the control algorithm. The main benefit of the prediction method is that one does not have to actually build the setup to quantify the trade-off in processor load and control performance for the event-driven controllers.

7

Conclusions and recommendations

7.1 Conclusions

Event-driven control is presented in the research hypothesis as a control design method to achieve a better overall system performance, compared to classical time-driven approaches, by relaxing one of the most stringent conditions that control engineers impose: a fixed sample frequency. System performance has to be understood in the sense of the combination of aspects that are influenced by the controller implementation. These are in particular: control performance (in terms of tracking, stabilization and disturbance rejection), software performance (in terms of processor load), amongst other aspects like communication bus load and system cost price.

In this thesis, we presented two particular event-driven control structures. The first one shows that by relaxing the equidistant sampling constraint, event-driven controllers can respond faster to changing conditions. The update of the proposed controller is triggered by new sensor data that comes available, which are the individual pulses of an encoder in the considered case. This means that the *exact* position measurement is used, instead of some estimation with a non-zero measurement error, which opens up the possibility to achieve high control performance, while operating with cheap, low resolution sensors. The controller tuning, for this fundamentally different controller compared to classical time-driven controllers, was performed by transforming the system equations from the time domain to the spatial domain. In the spatial domain, the encoder pulses, and therefore the controller triggering, occur equidistantly spaced. In this way, we are able to write the control problem as a synchronous problem such that variations on classical control theory can be applied to

design and tune the controller. The resulting control performance measures are also expressed in the spatial domain, in the sense that we obtain the bandwidth in the spatial frequency and aim at settling distances instead of (classical) settling times. When disturbances are also acting in the spatial domain - which is often the case - it can easily be determined how these disturbances are rejected. The proposed event-driven controller is experimentally validated in the printer where a one pulse per revolution encoder is used to accurately control the motion of images through the printer in the case study. By means of simulations and experiments on a prototype printer we show that with the event-driven controller a similar control performance can be achieved, compared to the originally applied observer-based controller in combination with a 12 pulse per revolution encoder. Furthermore, we showed that the processor load for the controller was reduced up to a factor 6.

The aim of the second proposed event-driven controller is to reduce the resource utilization (such as processor load and communication bus load) for the controller tasks by only updating the controller when necessary and not wasting computation time when there is no real need for it. For the particular presented application the controller was not updated when the tracking/stabilization error was below a certain threshold. By choosing this threshold, a trade-off is made between control performance and processing load. To get insight in this trade-off, theory is presented to analyze control performance in terms of ultimate bounds for the closed-loop system. The theory is based on inferring properties (like robust positive invariance, ultimate boundedness and convergence speed) for the event-driven controlled system from discrete-time linear systems (in case of "non-uniform sampling") or piecewise linear systems (in case of "uniform sampling"). Next to the theoretical analysis, simulations and experiments are carried out on a paper path test-setup. It is shown that for the particular application, the processor load was reduced by a factor 2 without significant influence on the control performance in comparison to a time-driven implementation.

To validate the potential of event-driven controllers for the processor load, a technique is presented to accurately estimate the processor load, prior to implementing the controller on a processing platform. This was done with relative low effort, despite the fact that many complex implementation factors are to be accounted for. For this purpose, micro measurements were used, together with simulation data to determine the number of control updates, to estimate the required processor load for the control algorithm.

From each of the presented examples of event-driven control, multiple trade-offs

become apparent. As this thesis focussed on the disciplines software and control engineering, these trade-offs involve control performance and software performance, amongst other system aspects like system cost price. One of the difficulties in system engineering discipline is to focus on the right, most important, trade-offs. For this purpose, the technique "treads of reasoning" was proposed and extended to identify the most important conflicts in the multi-disciplinary design of the paper flow control of the printer. This technique helps to structure in the typical chaos of uncertainty and the huge amount of realization options present in early design phases.

Threads of reasoning provides the system architect with valuable insight that supports him in making the important design trade-offs qualitatively. To quantify the design choices, simple models (like the models in section 4.8) are presented that capture the essence of the problem in the multiple domains. An open issue, however, is how to make the well balanced trade-offs based on these models. One solution is to use weighting functions, but the question then remains how to choose appropriate weights, as criteria often have different dimensions - "how to compare apples with pears" so to say. Therefore, in most design processes, the system architect is entrusted with this task. Although he should have the ability to reason over the multiple disciplines, it is a difficult job that is dominated by subjective arguments. Finding an approach that copes with this problem is an ongoing challenge of both industry and academia.

7.2 Recommendations

As event-driven control is a widely open and a barely explored field of research, many new applications can be considered that have great potentials for both industry and academia. The importance of this field is more and more recognized by industry, as they have to produce more complex systems for decreasing cost prices. This involves hard multi-disciplinary designs for which one cannot design controllers that only focus on the control performance, while posing hard demands on other systems aspects, like software implementation and sensor specifications.

7.2.1 Sensor-based event-driven control

This thesis proposes two examples of event-driven control, but the presented theory to analyze the controllers is certainly not limited to these specific examples. The analysis presented in chapter 4, could be applied to all kinds of applications in which sensors supply data that can be considered synchronous in the spatial domain. Examples are presented in chapter 3 (e.g. [16, 24, 56, 60]). Furthermore, examples are thought of in which sensor data is synchronous is another domain, like for instance temperature control, where the temperature is measured with a resolution of 0.1 *degrees*. The presented event-driven control design approach might have great potential for those systems as well.

In chapter 4 we have only presented one typical design of an event-driven controller implementation in the spatial domain. To be applicable in a much broader range of applications, more research has to be carried out for designing controllers in the spatial domain. An interesting question arises whether or not there are applications in which 'spatial' models are more natural than models with time as the independent variable. E.g. some of the disturbances in the printer are typically position dependent. One important area that automatically emerges is *spatial identification*. An interesting topic is how identification could be carried out by using low resolution sensors. Furthermore, it would be interesting to research how the spatial analysis could be incorporated in the design of other controller types, like H_∞, LQG and MPC.

7.2.2 Event-driven control to reduce resource utilization

The kind of analysis presented in chapter 5 could possibly be extended to analyze control over a network where package loss is involved. When a package is lost, the controller is in practice often implemented such that it holds the output, until new data

has arrived. This situation can be considered similar to the presented case of uniform sampling for the event-driven controller in chapter 5, where the output is held when the error is below a certain threshold. A difference with the situation analyzed in this thesis is that the loss of packages occurs *randomly* and is generally not coupled to a state of the system. An advantage in the case of package loss is that it is logical to assume that the time span, for which the control value cannot be updated, is limited. This guarantees the existence of a finite piecewise linear representation of the system for the presented analysis in the uniform case.

Although chapter 5 analyzes a particular event-driven control structure, it already indicates the complexity and challenge for the analysis and synthesis of these type of control loops for which the controller triggering cannot be considered synchronous in another domain. This work provides a first step towards a proper analysis of these types of control loops. Future work is to extend the theory with reference tracking. Moreover, to be applicable in industry, methods are required that can be applied to analyze control loops *within minutes*, as industry has to develop high-tech systems in very limited time spans.

Given the advantages of event-driven controllers and the various sources of event-triggering mechanisms present in industrial practice, it is fruitful to continue this line of research and to develop a mature event-based system theory.

Bibliography

[1] Adami, T.M., R. Sabala, and J.J. Zhu (2003). Time-varying notch filters for control of flexible structures and vehicles. In: *Proceedings of the Digital Avionics Systems Conference (DASC03)*, Vol. 2, pp. 7.C.2-1–7.C.2-6.

[2] Alexander, I. (2002). Towards automatic traceability in industrial practice. In: *Proceedings of the First International Workshop on Traceability*, Edinburgh, pp. 26–31.

[3] Altshuller, G. (2000). The innovation algorithm. TRIZ, systematic innovation and technical creativity. Technical innovation center, Worchester, Massachusetts, online: http://www.triz.org.

[4] Årzén, K.-E. (1999). A simple event-based PID controller. In: *Proceedings of the 14th World Congress of IFAC*. Beijing, P.R. China, Vol. 18, pp. 423–428.

[5] Årzén, K.-E., A. Cervin and D. Henriksson (2003). Resource constrained embedded control systems: Possibilities and research issues. In: *Proceedings of CERTS'03 - Co-design of Embedded Real-Time Systems Workshop*. Porto, Portugal.

[6] Balluchi, A., P. Murrieri and A.L. Sangiovanni-Vincentelli (2005). Controller synthesis on non-uniform and uncertain discrete-time domains. In: *Proceedings of Hybrid Systems: Computation and Control, Lecture Notes in Computer Science*, Vol. 3414, Springer Verlag, pp. 118–133.

[7] Baruah, S., D. Chen, and A. Mok. (1997). Jitter concerns in periodic task systems. In: *Proceedings of the Eighteenth Real-Time Systems Symposium*. San Francisco, CA, pp. 68–77.

[8] Bate, I., P. Nightingale and A. Cervin (2003). Establishing timing requirements and control attributes for control loops in real-time systems. In: *Proceedings of the 15th Euromicro Conference on Real-Time Systems*, Porto, Portugal.

[9] Bayer, J, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.M. DeBaud (1999). PuLSE: A methodology to develop software product lines. In: *Proceedings of the symposium on software reusability*, pp. 122–131.

[10] Berge, M.H., B. Orlic and J.F. Broenink (2006). Co-simulation of networked embedded control systems, a CSP-like process oriented approach. In: *Proceedings of the IEEE International Symposium on Computer-Aided Control Systems Design*, Munich, Germany, October 4–6.

[11] Blackman, S.S. (1986), Multiple target tracking with radar applications. Norwood, MA: Artech House.

[12] Blanchini, F. (1994). Ultimate boundedness control for uncertain discrete-time systems via set-induced Lyapunov functions. In: *IEEE Transactions on Automatic Control*, Vol. 39, No. 2, pp. 428–433.

[13] Blanchini, F. (1999). Set invariance in control. In: *Automatica*, Vol. 35, pp. 1747–1767.

[14] Bode, H. (1945), Network Analysis and Feedback Amplifier Design. New York: Van Nostrand Reinhold.

[15] Boyd, S., L. El Ghaoui, E. Feron and V. Balakrishnan (1994). Linear Matrix Inequalities in System and Control Theory. In: *Studies in Applied Mathematics*, Publisher: SIAM, Vol. 15.

[16] Bruin, D. de and P.P.J. van den Bosch (1998). Measurement of the lateral vehicle position with permanent magnets. In: *Proceedings of IFAC workshop on Intelligent Components for Vehicles*, Seville, Spain, pp. 9–14.

[17] Cervin, A., D. Henriksson, B. Lincoln, J. Eker, and K.-E. Årzén (2003). How does control timing affect performance? Analysis and simulation of timing using Jitterbug and TrueTime. In: *IEEE Control Systems Magazine*, Vol. 23, No. 3, pp. 16–30.

[18] Cloosterman, M., N. van der Wouw, W.P.M.H. Heemels and H. Nijmeijer (2006). Robust stability of networked control systems with time-varying network-induced Delays. In: *Proceedings of the IEEE Conference on Decision and Control*, San Diego, USA.

[19] Cockburn, A. (2000). Writing effective use cases. Addison-Wesley.

[20] Culler, D., J. Hill, P. Buonadonna, R. Szewczyk and A. Woo (2001). A network-centric approach to embedded software for tiny devices. In: *Proceedings of 1st International Workshop of Embedded Software*, Tahoe City, California, October 8–10.

[21] Dodds, S.J. (1981). Adaptive, high precision, satellite attitude control for microprocessor implementation. In: *Automatica*, Vol. 17, No. 4, pp. 563–573.

[22] Doff, R.C., M.C. Fatten and C.A. Phillips (1962). Adaptive sampling frequency for sampled-data control systems. In: *IRE Transactions on Automatic Control*, Vol. AC-7, pp. 38–47.

[23] ElGamal, A., C. Nair, B. Prabhakar, E.U. Biyikoglu and S. Zahedi (2002). Energy efficient scheduling of packet transmissions over wireless networks. In: *Proceedings of IEEE INFOCOM*, pp. 1773–1780.

[24] Förstner, D., and J. Lunze (2001). Discrete-event models of quantized systems for diagnosis. In: *International Journal of Control*, Vol. 74, No. 7, pp. 690–700.

[25] Franklin, G.F., J.D. Powell and M.L. Workman (1998). Digital Control of Dynamic Systems. *Third edition, MA: Addison-Wesley.*

[26] Franklin, G.F., J.D. Powell and A. Emami-Naeini (2005). Feedback Control of Dynamic Systems. *Fifth edition, Prentice Hall.*

[27] Freriks, H.J.M., W.P.M.H. Heemels, G.J. Muller and J.H. Sandee (2006). On the systematic use of budget-based design. In: *Proceedings of 16th annual international symposium of the INCOSE*, Orlando, Florida, USA.

[28] Freriks, H.J.M. (2005). White paper on designing with stepper motors. Online: http://www.esi.nl.

[29] Friedland, B. (1973). Optimum steady-state position and velocity estimation using sampled position data. In: *IEEE transactions on Aerospace and Electronic Systems.* Vol. AES-9, No. 6, pp. 906–911.

[30] Geer, D.(2005). Is it time for Clockless Chips? In: *Computer*, Vol. 38, No. 3, pp. 18–21.

[31] Glad, T. and L. Ljung (1984). Velocity estimation from irregular, noisy position measurements. In: *Proceedings of the IFAC 9th Triennial World Congress*, Budapest, No. 2, pp. 1069–1073.

[32] Goodman, J. (2005). Inroduction to fourier optics. 3rd ed., Robers & Company, 2005.

[33] Grewal, M.S. and A.P. Andrews (1993). Kalman filtering: theory and practice. Englewood Cliffs: Prentice Hall.

[34] Grieder, P. (2004). Efficient computation of feedback controllers for constrained systems. Ph.D. thesis ETH Zurich, Switzerland.

[35] Hayes, M.H. (1996). Statistical Digital Signal Processing and Modeling. New York: John Wiley & Sons.

[36] Heemels, W.P.M.H., R.J.A. Gorter, A. van Zijl, P.P.J. van den Bosch, S. Weiland, W.H.A. Hendrix and M.R. Vonder (1999). Asynchronous measurement and control: a case study on motor synchronization. In: *Control Engineering Practice*, Vol. 7, pp. 1467–1482.

[37] Heemels, W.P.M.H., B. de Schutter and A. Bemporad (2001). Equivalence of hybrid dynamical models. In: *Automatica*, Vol. 37, No. 7, pp. 1085–1091.

[38] Heemels, W.P.M.H., and J.H. Sandee (2006). Practical stability of perturbed event-driven controlled linear systems. In: *Proceedings of the American Control Conference*, Minneapolis, Minnesota, USA, pp. 4379–4386.

[39] Heemels, W.P.M.H., E. v.d. Waal, and G.J. Muller (2006). A multi-disciplinary and model-based design methodology for high-tech systems. In: *Proceedings of CSER*, Los Angeles, California, USA, April 7–8.

[40] Heemels, W.P.M.H., and J.H. Sandee (2006). Analysis of event-based controllers for linear systems. Submitted for journal publication.

[41] Henriksson, D. and A. Cervin (2005). Optimal on-line sampling period assignment for real-time control tasks based on plant state information. In: *Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference*, Seville, Spain, December 2005.

[42] INCOSE Technical Board (2004). Systems engineering handbook. A "what to"guide for all system engineering practitioners.

[43] Jazayeri, M., A. Ran, and F. vd Linden (2000). Software architecture for product families. Addison-Wesley Longman Publishing Co., Inc., Boston, MA.

[44] Jongeneel, C. (2005). Klokloze chips. In: *De Ingenieur*. Vol. 117, No. 4, pp. 52–53.

[45] Kao, C.-Y. and B. Lincoln (2004). Simple stability criteria for systems with time-varying delays. In: *Automatica*, Vol. 40, pp. 1429–1434.

[46] Kawka, P.A., and A.G. Alleyne (2005). Stability and Feedback Control of Wireless Network Systems. In: *Proceedings of the American Control Conference*, Portland, OR.

[47] Kerrigan, E. (2000). Robust Constraint Satisfaction: Invariant Sets and Predictive Control. Ph.D thesis. University of Cambridge.

[48] Kolmanovsky, I. and E.G. Gilbert (1998). Theory and computation of disturbance invariant sets for discrete-time linear systems. In: *Mathematical Problems in Engineering*, Vol. 4, pp. 317–367.

[49] Kopetz, H. (1993). Should responsive systems be event-triggered or time-triggered? In: *Transactions on Information and Systems*, Vol. E76-D, No. 11, pp. 1325–1332.

[50] Krucinski, M., Cloet, C., Tomizuka, M. and R. Horowitz (1998). Asynchronous Observer for a Copier Paper Path. In: *Proceedings of the 37th IEEE Conference on Decision and Control*, Tampa, Florida, Vol. 3, 1998, pp. 2611–12.

[51] Kvasnica, M., P. Grieder, M. Baotić and M. Morari (2004). Multi Parametric Toolbox (MPT). In: *Hybrid Systems: Computation and Control, Lecture Notes in Computer Science*, Vol. 2993, Springer Verslag, Philadelphia, USA, pp. 448–462, http://control.ee.ethz.ch/mpt.

[52] Lazar, M., and W.P.M.H. Heemels (2006). Global input-to-state stability and stabilization of discrete-time piece-wise affine systems. In: *Proceedings of the 2nd IFAC Conference on Analysis and Design of Hybrid Systems*, Alghero, Sardinia, Italy, June 7-9.

[53] Levis, P, S. Madden, D. Gay, J. Polastre, R. Szewczyk, A. Woo, E. Brewer and D. Culler (2004). The Emergence of Networking Abstractions and Techniques in TinyOS. In: *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation.*

[54] Lincoln, B. (2002). Jitter compensation in digital control systems. In: *Proceedings of the American Control Conference*, Anchorage, USA, pp. 2985–2990.

[55] Lindgärde, O. and B. Lennartson (1998). Comparing frequency analysis methods for sampled-data control. In: *Proceedings of the 37th Conference on Decision and Control*, Tampa, Florida USA. New York: IEEE, Vol. 1, pp. 829–834.

[56] Lunze, J. (2000). Diagnosis of quantized systems based on a timed discrete-event model. In: *IEEE Transactions on Systems, Man and Cybernetics*, Part A, Vol. 30, No. 3, pp. 322–335.

[57] Malan, R., and D. Bredemeyer (2005). Software architecture action guide. Online: http://www.bredemeyer.com.

[58] Muller, G.J. (2004). CAFCR: A multi-view method for embedded systems architecting; balancing genericity and specificity. Ph.D. thesis, Delft university of technology.

[59] Nechyba, M.C., and Y. Xu (1998). On discontinuous human control strategies. In: *Proceedings of the IEEE International Control of Robotics & Automation*, pp. 2237–2243.

[60] Phillips, A.M., and M. Tomizuka, (1995). Multirate estimation and control under time-varying data sampling with applications to information storage devices. In: *Proceedings of the 1995 American control conference*, Vol. 6, pp. 4151–4155.

[61] Ploplys, N., P. Kawka and A. Alleyne (2004). Closed-Loop Control over Wireless Networks. In: *IEEE Control Systems Magazine*, pp. 58–71.

[62] Putten, P.H.A. van der, and J.P.M. Voeten (1997). Specification of reactive hardware/software systems - The method software/hardware engineering. Ph.D. thesis, Eindhoven University of Technology, Eindhoven.

[63] Rakovic, S.V., P. Grieder, M. Kvasnica, D.Q. Mayne and M. Morari (2004). Computation of invariant sets for piecewise affine discrete time systems subject to bounded disturbances. In: *Proceedings 43rd IEEE Conference on Decision and Control*, pp. 1418–1423.

[64] Rakovic, S.V., E.C. Kerrigan, K.I. Kouramas and D.Q. Mayne (2005). Invariant approximations of the minimal robust positively Invariant set. In: *IEEE Transactions on Automatic Control*, Vol. 50, No. 3, pp. 406–410.

[65] ReVelle, J.B. (1998). The QFD handbook. Wiley.

[66] Robert, D., O. Sename and D. Simon (2005). Sampling period dependent RST controller used in control/scheduling co-design. In: *Proceedings of 16th IFAC World Congress*, Prague, Czech republic.

[67] López-Orozco, J.A., J.M. de la Cruz, E. Besada and P. Ruipérez (2000). An asynchronous, robust, and distributed multisensor fusion system for mobile robots. In: *The international Journal of Robotics Research*, Vol. 19, No. 10, pp. 914–932.

[68] Saaty, T.L. (1990). Multicriteria decision making: The Analytic Hierarchy Process. In: *RWS Publications*, Vol. 1, AHP Series.

[69] Saavedra-Barrera, R.H.(1992). CPU performance evaluation and execution time prediction using narrow spectrum benchmarking. Ph.D. Thesis, U.C. Berkeley, Technical Report No. UCB/CSD 92/684.

[70] Sandee, J.H, W.P.M.H. Heemels and P.P.J. van den Bosch (2005). Event-driven control as an opportunity in the multi-disciplinary development of embedded controllers. In: *Proceedings of the American Control Conference*, Portland, Oregon, USA, pp. 1776–1781.

[71] Sandee, J.H., P.M. Visser and W.P.M.H. Heemels (2006). Analysis and experimental validation of processor load for event-driven controllers. In: *Proceedings of the IEEE Conference on Control and Applications*, Munich, Germany, pp. 1879–1884.

[72] Sandee, J.H., W.P.M.H. Heemels, G.J. Muller, P.F.A. van den Bosch and M.H.G. Verhoef (2006). Threads of reasoning: A case study. In: *Proceedings of the 16th annual international symposium of INCOSE*, Orlando, Florida, USA.

[73] Sandee, J.H, W.P.M.H. Heemels and P.P.J. van den Bosch (2006). Analysis and experimental validation of an event-driven controller with a 1 pulse per revolution encoder. Submitted for journal publication.

[74] Sandee, J.H., P.M. Visser and W.P.M.H. Heemels (2006). Analysis and experimental validation of processor load for event-driven controllers. Submitted for journal publication.

[75] Sanz, R., and Årzén, K.-E. (2003). Trends in software and control. In: *IEEE Control Systems Magazine*, Vol. 23, No. 3, pp. 12–15.

[76] Schinkel, M., C. Wen-Hua, A. Rantzer (2002). Optimal control for systems with varying sampling rate. In: *Proceedings of the American Control Conference*, pp. 2979–2984.

[77] Schinkel, M., W.P.M.H. Heemels and A.Lj. Juloski (2003). State estimation for systems with varying sampling rate. In: *Proceedings 42nd IEEE Conference on Decision and Control*, pp. 391–392.

[78] Schiøler H., A.P. Ravn and J.D. Nielsen (2003). Impact of scheduling policies on control system performance. In: *Proceedings of EuroMicro Satelite Event on CoDesign in Real Time Systems*, Porto, Portugal.

[79] Sha, L., T. Abdelzaher, K.-E. Årzén, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, A.K. Mok (2004). Real time scheduling theory: a historical perspective. In: *Real-time systems*, Vol. 28, pp. 101–155.

[80] Sira-Ramirez, H. (1989). A geometric approach to pulsewidth modulated control in nonlinear dynamical systems. In: *IEEE Transactions on Automatic Control*, Vol. 34, No. 2, pp. 184–187.

[81] Sontag, E.D. (1981). Nonlinear regulation: The piecewise linear approach. In: *IEEE transactions on automatomatic control*, Vol. 26, pp. 346–357.

[82] Stark, W., H. Wang, A. Worthen, S. Lafortune and D. Teneketzis (2002). Low energy wireless communication network design. In: *IEEE Wireless Communication*, pp. 60–72.

[83] Veldhuizen, D.A. van, and G.B. Lamont (2000). Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. In: *Evolutionary Computation*, Vol. 8, No. 2, pp. 125-147.

[84] Vottis, C.V. (2003), Extracting more accurate position and velocity information from optical incremental encoders, SAI/2yr Thesis, Technische Universiteit Eindhoven, ISBN 90-444-0335-4.

[85] Wieringa, R. (2004). Requirements engineering: problem analysis and solution specification. ICWE, pp. 13–16.

[86] Zhang, W., M.S. Branicky and S.M. Philips (2001). Stability of networked control systems. In: *IEEE Control Systems Magazine*, Vol. 21, No. 1, pp. 84–99.

[87] Zitzler, E., and L. Thiele (1999). Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. In: *IEEE Transactions on Evolutionary Computation*, Vol. 3, No. 4, pp. 257–271.

[88] 20-Sim, modelling and simulation package (2006). Controllab Products Inc., Enschede, The Netherlands. [online] http://www.20sim.com.

Samenvatting

Regelsystemen zijn onmisbaar voor het goed functioneren van veel industriële high-tech systemen, zoals bijvoorbeeld kopieermachines en wafer steppers. In digitale regelsystemen worden tijd-continue signalen beschreven door middel van bemonstering op een veelal vaste frequentie, geïmplementeerd in een real time software omgeving. Als gevolg van het tijdgestuurde karakter van de regelsystemen stellen regeltechnici zware, niet bespreekbare eisen aan de real time implementatie van hun algoritmen, omdat op deze manier de vereiste regelprestaties gegarandeerd kunnen worden. Dit kan leiden tot niet optimale oplossingen als het ontwerpprobleem vanuit een breder multidisciplinair oogpunt wordt beschouwd. Een tijdgestuurde regelaar berekent bijvoorbeeld voortdurend nieuwe actuatorsignalen op een vaste frequentie, ook als er niets significant is veranderd in het proces. Dit is een onnodige verspilling van middelen als processor-rekentijd en communicatie-bandbreedte en is daarom niet optimaal als ook deze aspecten in beschouwing worden genomen.

Om de strenge real time eisen die de regeltechnici opleggen te verminderen, inclusief de daarbij behorende nadelen, stelt dit proefschrift voor om de strikte eisen van equidistant bemonsteren te laten vallen. Dit stelt de systeemontwerpers in staat om beter gebalanceerde multidisciplinaire afwegingen te maken en resulteert in een verbeterd systeemgedrag en in een gereduceerde kostprijs. Door geen equidistante bemonstering te eisen is het mogelijk om de bemonsterfrequentie te variëren en het uitvoeren van de regelalgoritmen dynamisch te plannen om zo de processor-rekentijd te optimaliseren. Ook is het mogelijk om de regelaar te activeren op het moment dat nieuwe meetdata is ontvangen. Op deze wijze kunnen reconstructiefouten, kwantisatie-effecten en vertragingen aanzienlijk worden verminderd, zodat de benodigde sensorresolutie, en daarmee de kostprijs, kunnen worden gereduceerd. Omdat in deze regelaars een gebeurtenis, of “event”, de regelaar activeert (bijvoorbeeld het ontvangen van nieuwe meetdata), in plaats van de tijd, noemen we dit type regelaars gebeurtenisgestuurd, oftewel “event-driven”.

In dit proefschrift worden twee verschillende event-driven regelaarstructuren behandeld. De eerste is een sensor-gestuurde event-driven regeling, waar de regelaar wordt geactiveerd door het ontvangen van nieuwe sensordata. Deze regelaarstructuur is gebruikt voor het nauwkeurig sturen van een motor, op basis van een (extreem) lage

encoderresolutie. Het regelaarontwerp is gebaseerd op een transformatie van de systeemvergelijkingen in het tijddomein naar het hoekpositiedomein (spatiële domein). Omdat de regelaaraansturing synchroon is ten opzichte van de hoekpositie van de motor, kan de klassieke regeltheorie worden toegepast voor het ontwerpen en instellen van de regelaar. Door de transformatie worden ook de regelkarakteristieken, afkomstig uit de systeemanalyse, geformuleerd in het spatiële domein. De bandbreedte van de regelaar wordt niet meer uitgedrukt in Hertz (s^{-1}), maar in rad^{-1} en de *tijd* waarin het systeem tot rust komt wordt vervangen door een *afstand* waaronder rust wordt bereikt. Deze spatiële maten relateren in veel high-tech systemen direct aan de echte eisen aan de prestatie. Bovendien kunnen verstoringen vaak eenvoudiger worden geformuleerd als functie van de positie dan als functie van de tijd. Ter validatie van de theorie is de voorgestelde regelaar geïmplementeerd in een systeem dat met hoge snelheden documenten print. De regelaar stuurt nauwkeurig een motor aan op basis van een encoder met een resolutie van slechts 1 puls per omwenteling. Middels analyse, simulatie en metingen wordt aangetoond dat de regelaarprestaties vergelijkbaar zijn met de initieel toegepaste industriële regelaar die is gebaseerd op een veel hogere encoderresolutie. Bovendien vraagt de voorgestelde event-driven regelaar een significant kleinere processorbelasting. Vanuit systeemperspectief kan geconcludeerd worden dat deze regelaar de tijdgestuurde regelaar overtreft.

Het tweede type event-driven regelaar heeft als specifieke doel de processorbelasting en communicatie-bandbreedte voor de regelaar implementatie te reduceren. De regelaar wordt alleen aangestuurd als het werkelijk noodzakelijk is. Als voorbeeld wordt een regelaar gepresenteerd die alleen wordt geactiveerd als de volg- of stabilisatiefout groter is dan een bepaalde drempelwaarde. Door deze drempelwaarde te kiezen wordt er een directe afweging gemaakt tussen de regelaarprestaties en de processorbelasting. Om meer inzicht in deze afweging te krijgen, wordt een theorie gepresenteerd om de regelaarprestaties te analyseren die worden uitgedrukt in maximale begrenzingen, “ultimate bounds”, van de volg- of stabilisatiefout. De theorie is gebaseerd op afgeleide eigenschappen (zoals robuuste positieve invariantie, “ultimate boundedness” en convergentie indices) van het event-driven bestuurde systeem vanuit tijd-discrete lineaire systemen en stuksgewijs lineaire systemen. Naast de theoretische analyse zijn simulaties en experimenten uitgevoerd op het papierpad van een printer testopstelling. Voor de specifieke opstelling is aangetoond dat de processorbelasting gereduceerd is met een factor 2, ten opzichte van een tijdgestuurde implementatie, zonder significante achteruitgang van de regelaarprestaties. Bovendien is er een methode

ontwikkeld om de processorbelasting nauwkeurig te voorspellen voor verschillende processoren. De methode is gebaseerd op simulatiemodellen en micro-metingen op de processor, zodat de processorbelasting voorspeld kan worden voordat het regel algoritme is geïmplementeerd.

Naast deze bijdragen op het gebied van event-driven regelaars is de systeem engineering techniek "threads of reasoning" uitgebreid en toegepast op het printerontwerp om als ontwerper te focussen op de juiste ontwerpproblemen en -afwegingen.

Samenvattend zijn er twee event-driven regelaars theoretisch geanalyseerd en experimenteel gevalideerd op een prototype high-tech print systeem. De resultaten illustreren de potentiële voordelen van event-driven regelen met betrekking tot het totale systeemgedrag en voor het maken van afwegingen tussen regelaarprestatie, softwareinspanning en kostprijs.

Acknowledgements / Dankwoord

Wat een heerlijk gevoel om na het afronden van dit proefschrift een dankwoord te mogen schrijven aan alle mensen die me in de afgelopen vier jaar hebben geholpen, in wat voor vorm dan ook. Voordat ik enkele mensen persoonlijk bedank, wil ik graag laten weten dat ik een mooie en leerzame tijd achter de rug heb, waaraan iedereen in mijn omgeving zijn of haar steentje heeft bijgedragen.

Als eerste wil ik mijn copromotor Maurice Heemels bedanken voor de intensieve support en samenwerking. Met jouw creatieve inzichten en gedegen wiskundige kennis is het werk vele malen sterker dan waar ik in mijn eentje ooit had kunnen komen. Ook promotor Paul van den Bosch heeft hier een belangrijke rol in gespeeld door het altijd zien van nieuwe mogelijkheden en het concretiseren van de werkelijke waarde van het werk. Verder ben ik Paul zeer dankbaar voor de mogelijkheid om drie maanden van mijn onderzoek uit te voeren aan de universiteit van California, Berkeley, in de Verenigde Staten. Hierbij wil ik tevens mijn gastheer aldaar, prof. Tomizuka, en zijn groep bedanken voor de gastvrijheid.

Op deze plaats wil ik ook de leden van de kerncommissie, Maarten Steinbuch, Job van Amerongen en Dinesh Verma, hartelijk bedanken voor het doornemen van het promotiewerk en voor het waardevolle commentaar en suggesties. Special thanks to Dinesh for reading the thesis just before and after his honeymoon.

Mijn promotie heb ik uitgevoerd binnen het Boderc project van het Embedded Systems Institute (ESI). Bij deze wil ik dan ook alle medewerkers van het ESI bedanken voor de prettige samenwerking. Speciaal dank aan Frans Beenker en Gerrit Muller voor de ondersteunende gesprekken. Ook wil ik alle leden van het Boderc project, en met name collega-promovendi Björn, Peter, Marieke, Marcel en Oana bedanken voor het vele nuttige teamwerk. Zeker de lol die ik met Björn heb gehad tijdens diverse conferenties zal me nog lang bijblijven. De samenwerking met onze tukker Peter is een belangrijke bijdrage van dit werk geworden. Graag wil ik jullie veel succes toewensen bij het afronden van de promotie. Binnen het Boderc project heb ik verder veelvuldig en zeer prettig samengewerkt met Océ medewerkers: Peter van den Bosch, Hennie Freriks en Jos Nelissen. Samen hebben we heel wat “modellen werk” gedaan. Voor de succesvolle implementatie van een controller op een prototype Océ printer wil ik Sander Hulsenboom, Lars Idema en René Waarsing bedanken voor hun hulp en cre-

atief meedenken. De industriële relevantie van het werk die daarmee is aangetoond was voor mij een van de grootste drijfveren.

Twee dagen in de week werkte ik tussen mijn grote vrienden van de Control Systems groep van de faculteit Elektrotechniek. Aan de vele pauzes, Benelux meetings en uitstapjes heb ik veel plezier beleefd, bovenal met collega-promovendi: John, Andrej, Nelis, Mark, Mircea, Maarten, Satya, Michal, Michiel, Aleksandar, Patricia, Leo, Bart and Mario. Heel wat white-boards heb ik in deze tijd weten vol te kleuren met kamergenoot John Kessels; van control schema's tot bouwkundige installaties, van wiskundige formules tot artistieke uitingen.

Voor de noodzakelijke afleiding tussen het promoveren door bedank ik de Eurobotters waarmee ik de laatste 2 jaar heel wat avonden, weekenden en nachtjes op de TU doorgebracht heb. In het bijzonder Frank van Heesch, die ook nog eens verantwoordelijk is voor het ontwerp van de kaft van dit proefschrift en oud-huisgenoot Koen Willems, waarmee ik grote delen van mijn promotiewerk aan de eettafel heb doorgesproken.

Als laatste, een speciaal woord van dank voor mijn grote liefde Mijntje en mijn ouders en familie. Bedankt voor alle steun en liefde die jullie mij hebben gegeven tijdens deze drukke periode.

Eindhoven, november 2006

About the author

Heico Sandee was born in 1978 in Kamperland, The Netherlands. He received his Master of Science (MSc) degree in Electrical Engineering from the Eindhoven University of Technology (TU/e) in August 2002. He carried out his MSc project at the company Vanderlande Industries, Veghel, The Netherlands. This project was about the "Investigation into the use of an asynchronous linear induction motor as sensor for its control".

After receiving his MSc degree, Heico worked for three months at the TU/e in the project "The development of an energy and power management system for conventional and future vehicle power nets", in cooperation with Ford Research, Aachen, Germany.

From January 2003 to December 2006 he pursued his PhD degree in the Control Systems group of the faculty of Electrical Engineering, TU/e. The project he was working on is called Boderc (Beyond the Ordinary: Design of Embedded Real-time Control), coordinated by the Embedded Systems Institute. The project focused on the multidisciplinary design, analysis and validation of embedded dynamical systems. His main research interest is real-time control applications, and more specifically the application of event-driven controllers, as presented in this thesis. During this period he worked closely together with Océ Technologies B.V., Venlo, The Netherlands.

In 2005, he successfully finished the DISC (Dutch Institute of Systems and Control) course program. In this year he also visited the Mechanical Systems Control Laboratory at UC Berkeley, California, USA, for which he received an NWO scholarship for three months.

Heico's personal interests are music, (in particular drums and vocals), sports (mountain biking, running) and robotics (participation in Eurobot 2001, Robotwars 2002, TNO Robotcompetition 2004, Eurobot 2004, 2005).