Eindhoven University of Technology
Department of Electrical Engineering
Control Systems

TU/e technische universiteit eindhoven

Analysis of varying sampling frequency in controller algorithms

by Maurice Snoeren

Master of Science thesis

Project period: March 2004 – May 2005
Report Number: 05A/03
Commissioned by: Prof. dr. ir. P.P.J. van den Bosch

Supervisor:
Ir. J.H. Sandee

Additional Commission members:
Prof.dr. ir. P.P.J. van den Bosch
Dr. ir. A.A.H. Damen
Dr. ir.W.P.M.H. Heemels

TO MY DAD,

PREFACE

Before I started with my graduation project, I did not know that there were many difficulties with the design of embedded control systems. I was always interested in these kind of systems, because of the combination of electronics and software; especially robotics. This makes these systems very complex and very dynamical. It is namely possible to change the (controller) behaviour very extremely only by changing the software code. However, this flexibility results also in many difficulties with the design of embedded control systems. The software, electronics and mechanics cannot be easily matched and modelled as a total embedded system and it needs more research. This makes the world of embedded control systems a nice and interesting playing ground.

The project assignment was related to these kind of problems and gave me the opportunity to study these complex difficulties that rise with the implementation of controllers on software platforms of embedded systems. My passion for embedded control systems has grown and will be always of special interest to me. Therefore, I want to thank prof. P.P.J. van den Bosch and Heico Sandee for the opportunity to do research to an embedded control system problem at the Measurement and Control group. With Heico I had a lot of nice discussions about embedded systems, robotics and off course my project assignment and that was very valuable to me.

During the project it was very cosy to work in the graduation room. Every day I was surprised that the time was going really fast. Everyone's graduation assignment and off course social things were discussed and that resulted into a comfortable working place. I want to thank all the members, the Ph.D. students and my colleague graduation students at the Measurement and Control group for the nice time.

A lot is happened, last year, during my graduation project and now a new period will start. I want to thank especially Jeske for all the support during my study period. Without you, I would not be where I am now today. Thank you. At last I want to thank all my friends that have supported me and offered themselves to listen to my great stories about my graduation project.

ABSTRACT

Controller systems become more and more complex from the perspective of both control and computer science. More often, controllers are implemented on multitasking realtime computing platforms. The software engineer cannot guarantee a deterministic behaviour of the computer system and that complicates the implementation of controller algorithms on embedded systems. Randomly variable delays are introduced by the software and the controller algorithm cannot deal with that, because it needs mostly strong timing requirements. There exists a gap between the control and software engineering with the design of embedded systems. However, when we want to design an optimal embedded controller system, cooperation between these communities is needed.

We can distinguishing three delays that occur in a controller algorithm: measurement, computation and actuator update delay. Computational delay is assumed to be constant and measurement and actuator delays vary with a randomly fashion. Controller systems with variable delays in the loop become time-variant and all existing time-invariant controller theory cannot be used. The study starts with a global analysis of a controller system, with which we get the sensitivity to very small sampling period variations of the transfer functions. To analyse variable delays this theory is not applicable and we need other methods. We present a model in which the embedded controller system can be described with time-invariant transfer functions. The influence of the variable delays is fed into the system by using two extra disturbance inputs. In fact, the disturbance signals represent the error between the system with and without the variable delayed actions. Analysis of the disturbance signals show the influence that the variable delays have on the controller system. The model can be used to simulate a sampled-data system, which is exposed to variable delays, in the time domain. In the frequency domain we see that the disturbance inputs are time-varying and cannot be described by a transfer function. Because the disturbances depend on the signals of the control system, complex loops are introduced. These time-varying loops have to be considered with the description of the transfer function of the model as well.

Feedback scheduling will play an enormous role in the design of embedded systems. This method uses a feedback loop from the controller algorithm to the software scheduler and visa versa. It will connect the control algorithm to the software. The controller and the scheduler can dynamically adjust parameters of the embedded controller system in order to suppress errors that are caused by variable delays.

Samenvatting

Vanuit het perspectief van de regel- en de softwaretheorie worden regelsystemen steeds complexer. Regelaars worden steeds vaker op multitasking computer systemen geïmplementeerd. Hierdoor kan het deterministisch gedrag van het computerplatform niet meer gegarandeerd worden. Er worden variabele vertragingen geïntroduceerd in het systeem waardoor er problemen ontstaan met de implementatie van regelaars in embedded regelsystemen. Op dit moment eisen regelalgoritmen perfecte en equidistante bemonsteringstijden en kan er nog niet worden omgegaan met variabele vertragingen. Tussen de regel- en de software-ingenieurs bestaat er een groot gat als het gaat om het ontwikkelen van embedded systemen. De ontwikkeling van een “optimaal” embedded regelsysteem vergt wel een goede en nauwe samenwerking van beide vakgroepen.

We kunnen drie vertragingen onderscheiden binnen een regelalgoritme: bemonster, calculatie en actuator update vertraging. We nemen aan dat de berekening van het algoritme een constante vertraging veroorzaakt. De bemonster en actuator update vertraging varieert in iedere bemonsterperiode en krijgt telkens een willekeurige waarde. Wanneer de regellus vertragingen bevat, wordt het regelsysteem tijdvariant en kan de bestaande regeltheorie voor tijd-invariante systemen niet meer gebruikt worden. De studie begint met een analyse waarbij we globaal de gevoeligheid, voor kleine variaties in de bemonsteringstijd, van de verschillende overdrachten van het systeem bekijken. Voor het analyseren van variabele en grotere vertragingen is deze theorie niet toepasbaar en moeten we naar andere methoden zoeken. Een model wordt beschreven waarmee we het regelsysteem kunnen beschrijven met bestaande tijd-invariante methoden. De invloed dat de variabele vertragingen hebben op het systeem, wordt via verstoringen terug in het systeem gebracht. In principe representeren de signalen van de verstoringen de fout dat gemaakt wordt tussen het systeem met en zonder variabele vertragingen. Het model kan goed gebruikt worden om een sampled-data regelsysteem, met variabele vertragingen in de regellus, te simuleren in het tijddomein. Door de variabele vertragingen worden de signalen van de verstoringen zijn tijd-variant. Omdat de verstoringen afhankelijk zijn van het regelsysteem, bestaan er twee complexe lussen. Deze overdrachten kunnen niet worden beschreven met een tijd-invariante overdrachtsfuncties. Bij het bepalen van de totale overdracht van het model, dienen deze complexe tijd-variante lussen meegenomen te worden. Analyse in het frequentie domein is hierdoor niet eenvoudig.

In embedded systemen wordt “feedback scheduling” (plannen) van regel taken erg belangrijk. Met deze methode ontstaat er een lus tussen de software en het regelalgoritme. Hierdoor wordt de software en de regelaar met elkaar verbonden. De fout, die ontstaat door de variabele vertragingen, kan dan worden weggeregeld. Het systeem kan hierdoor beter anticiperen op de geïntroduceerde dynamische vertragingen.

TABLE OF CONTENTS

1. INTRODUCTION

Controller systems become more and more complex from the perspective of both control and computer science. Hardware devices for a network and a software platform become cheaper. Often, we see that even the "simplest" embedded control system, contain a multitasking realtime operating system. There is also the possibility to control over a communication network; getting sensor data and sending data to actuators. For optimal use of the computing resources, the control algorithm and the software design, need to be considered at the same time. Because of the fact that the control and the software science focus on their own problem domains, many difficulties arise with the design of embedded motion control systems. Many computing platforms are not able to give any deterministic guarantees. Therefore, different delays are introduced in the different states of the controller algorithm that is executed on the processor. This variable delay can have a major impact on the stability and the performance of the plant under control. The introduced delay variations in the controller system, makes the system time-varying. All theory for analysis and design of time-invariant systems cannot be used directly, so new analysis methods are needed to get more grip on these systems.

The control engineers often assume that the software platform, that is used to implement the controller algorithm, can provide deterministic and fixed sampling periods as needed. On the other hand, the software engineers generally assume that all control algorithms can be modelled as periodic tasks with constant periods, hard deadlines and known worst-case execution times. It is quite clear that these assumptions are not necessarily true. Furthermore, the software engineer is not aware of the performance and stability degradation when the software implementation does not meet the timing requirements. There exists a gap between these two communities. In the embedded control system design, we need more cooperation between the control and software community.

Many problems rise with the design of embedded control systems. Generally, control engineers assume that there is enough computing power on the platforms to calculate their algorithms. Of course that is certainly true. However, the trend nowadays is to utilize the computing platform at its maximum. Not only controller algorithms, but also a lot of other tasks, can be running on the same processor. This implies that we want the lowest possible sampling frequency for our controller algorithms to get less load on the software system. Rules of thumb[1] are in this case to conservative and we want to get more grip and analysis to determine the sampling frequency of a particular controller algorithm [7].

Embedded control systems can be very complex and there is not much research available. Due to many problems that we have with these type of systems, we have to understand the operation of these complex systems more better. What kind of delay variations are introduced in the controller algorithm? Can we get the sensitivity to sampling period variations of a particular system? What can we do, to connect the controller and the software platform that it becomes more robust to introduced variations? Describing these systems with existing theory, is that possible? Is it possible to determine the effect of the system, that is caused by the delay variations? Can we make existing or new controller algorithms in a simple way more robust to varying delays? We see that a lot of study can be done in the field of embedded control systems. These questions represent the main study that is done and is described in this report. It gives an analysis that gives more understanding of these systems. This study considers only delay variations that are caused by the software platform, such as measurement, computational and actuator update delays.

We start with chapter 2 and discuss embedded control systems. What kind of embedded systems are we considering and how it is globally implemented. Chapter 3 describes different definitions concerning delay variations. When analysing delays of actions it is practical to define the

1 It is common to choose the sampling frequency twenty times the bandwidth of the system.

behaviour different actions can have. This can be helpful when we restrict delays to some definition, so the equations in some cases can be simplified. Chapter 4 discusses how delays normally are described in the controller theory. Also two other approaches to describe varying delays are described. The first analysis to get the global sensitivity to delay variations is described in chapter 5. Here we set-up an analysis based on the sampled-data method. Chapter 6 describes the model which is developed that can describe the controller system with linear time-invariant equations. The system is externally exposed to delay variations. Sampled-data systems are mostly analysed in the frequency domain. Chapter 7 describes the model of chapter 6, but transforms the equations to the frequency domain. Simulation based on the model gives more insight of the behaviour of the real system. This is described in chapter 8. Analysis in the time domain is done in chapter 9. With the knowledge of the model and the description of the delay variation signal contents, we can suggest different approaches to improve embedded controller algorithms. These different ideas are discussed in chapter 10. Conclusions of the graduation work are given in chapter 11.

2. EMBEDDED CONTROL SYSTEMS

Embedded systems are complex systems, because they consist of different technologies that are internally connected and communicate with each other. Mostly, software is a characteristic part of an embedded system. There are a lot of embedded systems already developed. Just think of your mobile phone, washing machine, microwave, etc. Nowadays in almost every device a micro-processor is implemented, because of their extremely low cost price. The main advantage to use software, is that changes easily are made and that makes the system design dynamical and flexible.

We consider in this report only embedded motion control systems. These systems consist of three parts, namely software, electronics and mechanics. We can define the information flow between these parts. As we see in figure 2.1 the software is connected with the electronics and the electronics is connected with the mechanics. On the software platform different tasks are running. For example, a controller task that controls the speed of a motor and a controller task that controls a robot arm. These controller algorithms have to meet strong timing requirements to guarantee stability and performance of the plant under control. Besides the controller algorithms other tasks are running that have to meet less timing requirements. In this example, a user-interface application handles the user interaction via the keyboard and display, and a network application handles the network traffic to and from the system. Physically, the user-interface task communicates with the keyboard and display electronics and the network task with the network adapter. Actuators consist mostly of both electronics and mechanics. This is also shown in figure 2.1. The controller algorithms communicate with the connected sensors and actuators of the system. For example a motor converts an electronic current to a mechanical torque. In this way the electronics communicates with the mechanics. The motor or robot arm motion is controlled by the software via the electronics.

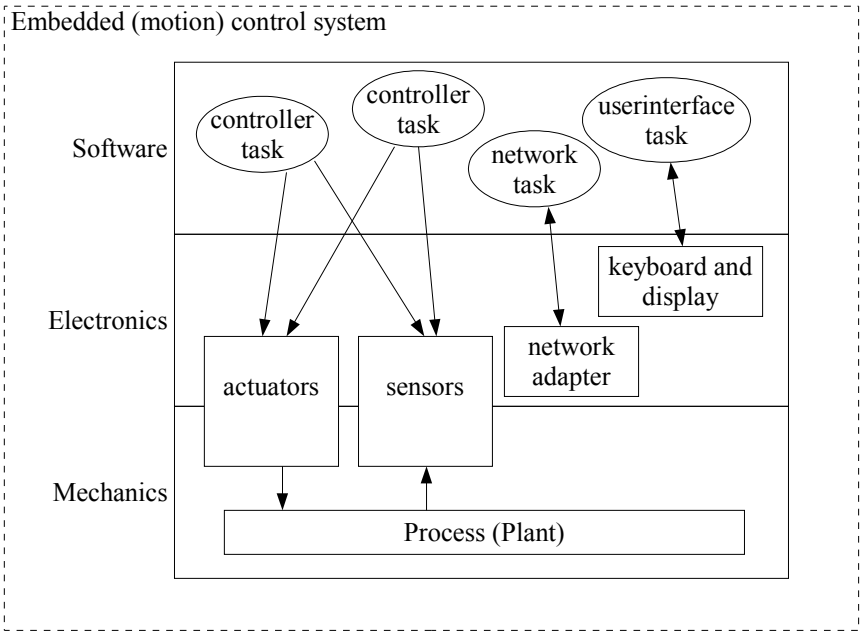

Figure 2.1 Structure of a complex embedded control system.

Each part of the embedded system is a science on its own and uses different methods and theories. For example controller theory uses dynamical differential equations. With the z-transform we are able to describe the controller in discrete time and therefore it can be implemented in software as a periodic task. Software code consists of a collection of statements that is compiled into assembler code that is executed by a processor. The definition of the statements, that can be

used, is dependent on the used programming language. A lot different types of the programming languages are developed. Often controller implementation is done in assembler code and more and more we see also implementations with C.

However, we note that these parts cannot simply be connected. Each part can be analysed independently, but there are a lot of problems to analyse the behaviour of the total embedded system behaviour. In fact, we are not able to say anything about the total behaviour in the design, but only when we have developed a prototype of the system.

Because of the complexity, we see that there exist many dependencies among each part (see figure 2.2). This means that when a change is made in one part of the embedded system, it directly effects the other two parts as well. The influence of these changes do not have to be trivial always. For example, software engineers assume that it can be afforded to relax the timing requirements a bit of the scheduling of the tasks. Causing, that the measurement samples are not always exactly on the moments that they have to be taken. This directly has effect on the performance and stability of the plant under control (mechanics), because the controller algorithm may need these strong timing requirements. This makes the analysis of the system even more complex. Much dependencies are not known and cause implementation problems. Approaching such systems needs all the different communities during embedded system design together. A method with which we are able to describe the behaviour and design of the total embedded system is very important and needs research[2].

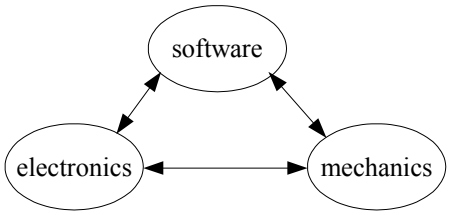

Figure 2.2 An embedded system contains different parts. There exist many dependencies between each part (defined by the arrows), the system becomes very complex..

2.1. CONTROLLER THEORY

Motion controller algorithms are more and more implemented on a processor, where both hard-realtime (HRT) and soft-realtime (SRT) tasks are executed. Hard-realtime tasks mostly have strong timing requirements that need to be fulfilled by the software platform. Soft-realtime does not have strong timing requirements. Many different control theories and methods are developed that deal with discrete time control systems. However, this theory assumes that the sampling (analogue-to-digital conversion), the execution of the code and the update of the actuator (digital-to-analogue conversion) takes no time and is repeated perfectly equidistant in time. In the real situation delays are introduced between these three steps and are often neglected in the analysis. Only fixed delay time, for example a worst-case computation time, is considered in the design when analysis comes closer to practice. The determination of the sampling rate of a controller algorithm is mostly done by some “rule of thumb”. In the case of continuous time design, transforming the continuous time controller to a discrete time controller, the sampling frequency is chosen twenty times of the systems bandwidth [14]. Only then the behaviour of the discrete time controller approaches the continuous time controller. Other controller design methods, for example direct z-transform, in which the plant is transformed to its discrete equivalent, choses the sampling frequency six times the systems bandwidth [14]. We see that the sampling frequency can differ a lot and that can be very important in the software platform design. For example, when the controller algorithm does not use

2 Modelling the total embedded system design is the main research of the Boderc project, which is started by the TU/e and OCÉ .

much computing resources, more tasks can be executed or a slower processor can be used. Determination of the best sampling frequency has to investigated with the design.

2.2. SOFTWARE IMPLEMENTATION PROBLEM

When we design motion controller algorithms for embedded system implementations, two disciplines are concerned; control and software. The control community focusses on its own problem domain, such as controller and observer design. Also the software community focus on their own problem domain, for example scheduling, programming languages and operating systems. This causes a gab between these two disciplines in the design of embedded systems. To tackle the implementation problem, it becomes very important to consider both control and software design at the same time. To obtain the optimal solution we have to use all resources that exist in both domains during the design.

A controller for a specific application is normally designed according to specified requirements. In spite of the fact that the controller will be implemented on a software platform, the total design of the controller is done by the control engineer. When the controller is developed, frequently the assumption is made that this controller can easily be implemented in the software. But because of the strong timing requirements that come with the control algorithm, such as a very accurate and high sampling period, implementation can be very hard and becomes more often a problem. During the design there is barely no communication between the control and the software engineers, while both disciplines are concerned with the problem.

2.3. SCHEDULING

Controller algorithms often need strong timing requirements to guarantee stability and performance of the plant under control. To meet these timing requirements the controller algorithm has to get immediately the required computing capacity when that is needed. Many computing platforms cannot guarantee any deterministic behaviour and that increases the complexity of the timing. Mostly, this is caused by elements that increase the mean performance of the software platform. Implying that the worst-case performance of the platform becomes worse. An example of such element is the cache. This element is a very fast memory which is directly attached to the processor, such that the processor can process the data very fast. The cache autonomously fill its memory that the processor needs in the nearby future, so it can be that the memory is filled with wrong data and that costs time. Generally, the mean performance of the computing system will increase and the worst-case performance decrease. The determinism of the software platform will degenerate with the use of such elements. However, this is not feasible when running control algorithms that need worst-case performance, so we want to avoid these elements or learn to deal with it in the future.

With scheduling algorithms we are able to time each task that is executed on the software platform. This is a way to fulfil the timing requirements of the controller algorithms. Unfortunately, it is very difficult to predict when a certain task is initiated on a multitasking platform. However, the execution time of a controller algorithm is very constant and well to predict. Research is done to predict the timing of different tasks on software platforms. For example, the language POOSL (Parallel Object-Oriented Specification Language) can capture the functional behaviour of a system [11]. Timing of the executed tasks can be obtained or predicted and is used in these models. This way, the timing behaviour of the system can be analysed. Knowing the time-deviation that exists between the model and its implementation is very important to predict the behaviour of the implemented system [12]. It can be decreased or increased by using different hardware devices in the software platform. This time-deviation can be used in the controller and scheduling design to create a more robust embedded system to these delay variations.

Interesting research in the software domain concerning controller implementation are scheduling, programming languages and operating systems. These are sequentially discussed in this section. Two important methods, used for realtime scheduling, are the Rate-Monotonic Analysis (RMA) and Earliest Deadline First (EDF) algorithm [5]. These methods provide analysis and algorithms with which we can specify, understand, analyse and predict timing behaviour of realtime software systems. With EDF it is even possible to negotiate with the scheduler about a particular task. The scheduler calculates whether or not the task can be executed, based on its timing requirements and the schedule of the already running tasks on the platform. This gives opportunities for online feedback about the load of the running system.

Many programming languages are developed to implement faster and easier complex control structures into the software. But when the abstraction level increases, we normally see that the optimisation of the source code decreases. This implies also a decrease of performance when it will be executed. Even the present controllers are often implemented with assembler code, which makes the application platform dependent. After all, it becomes tough to keep up with the recent developed programming techniques. We see more and more that also low level programming languages, like C, are used to implement controller algorithms. When we have to create software code that has to meet strong timing requirements, several (realtime) programming languages can be used. Two examples are the PCL (Program Control Language) and Esterel (Synchronous programming language). The main property of these languages is that they can deal with the real time. So the timing of the code can be described by statements. This is usually not the case with most standard programming languages, like C.

There are different types of operating systems (OS) that we can use. We are familiar with a few OS that we also use on our personal computer, for example Windows or Linux. Because the standard OS are not able to guarantee realtime timing specification, there are special, for this kind of systems, realtime OS developed. For example, we have Windows Realtime and Linux Realtime. These realtime OS mostly provide execution timing guarantees. In this way, the system becomes more deterministic on the timing behaviour of the tasks that run on the software platform.

2.4. EMBEDDED CONTROLLERS OF THE FUTURE

We see that there is a gap between the software and the control with the design of embedded control systems. The design of a new generation of embedded controllers require the cooperation between these two communities. Problems that rise in the controller design can be reconsidered in the software design and vice versa. Combining these two domains offers a lot more possibilities and methods to design eventually the optimal controller, software platform and scheduler.

The main implementation problem of controllers on embedded systems, that the software engineers have, are the strong timing requirements that come with the controller design. Focussing on the controller design, we can analyse the possibility to stretch these timing requirements. Moreover, we have to design controllers that are robust to these kind of timing variations. The conclusion of paper [12] is that it is possible to determine the worst-case timing delay and can be used in the embedded controller design.

We can also analyse the problems from a software perspective. The scheduling algorithms can be divided into two categories: static and dynamic scheduling. Static scheduling is commonly used and schedules the tasks based on predefined knowledge about these tasks. For more demanding applications, that have to provide flexibility with limited computing power, dynamical scheduling can be an improved approach to schedule controller tasks. This scheduling is based on feedback from the controller to the scheduler and feedback from the scheduler to the controller. Actually, we create a control loop between the scheduler and the control algorithm and we are able to adjust the behaviour of the system during execution. Two papers [10] and [22] discuss such feedback

scheduling with promising results, so scheduling methods become more aware of the performance of the control algorithm that is executed and vice versa and can be an advantage for embedded control. However, the overhead of the scheduling algorithm is not well discussed and needs computing resources as well.

There are many possibilities with the design of embedded control systems. For example, even statistical multiplexing can be applied. So, when there are many control tasks running, the chance that every control algorithm needs its maximum computing capacity on the same moment is very small. However, a lot of research is to be done in this field. Combining software and control will open a whole new world to embedded control systems.

3. DEFINITIONS CONCERNING VARIABLE DELAYS

Before we are able to analyse embedded controller systems, with variable delays in the loop, we have to define the actions of the controller and different properties that a certain delayed action can have. Section 3.1 describes the actions that occur in a controller algorithm and introduces with each action a delay variable. In section 3.2 we discuss how we can define different types of delayed actions.

3.1. CONTROLLER SYSTEM ACTIONS

We consider control systems in which the controller is discrete and is implemented on a software platform. Even the simplest embedded systems contain a multitasking operating system. Figure 3.1 shows a simple control scheme in which the delays and actions that occur in the software are displayed in the dashed box. We see three delays in this figure: measurement, computation and actuator update delay. We consider controller algorithms that sequentially read the measurements,

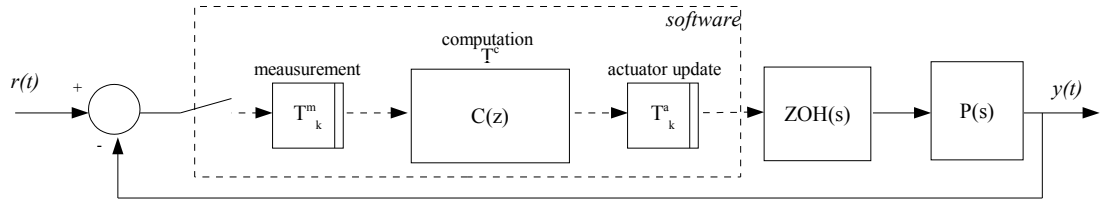

Figure 3.1 Simple controller structure. The blocks before and after the controller C(z) represent the delay variations that occur in software when an action is taken. T^m_k represents the delay of the measurement and T^a_k the delay of the actuator update. Computation time of the algorithm is assumed constant and is therefore defined by T^c.

compute the output and update the actuator value. Due to the fact that the software cannot guarantee any deterministic timing behaviour of the platform, delay variations in these algorithm steps are introduced. These variable delays, cause that the measurement and actuator updates are not equidistantly spaced in time. Therefore, two delay variations T^m_k and T^u_k are introduced, which respectively represent the measurement and actuator update delay at time k. In general, the computation delay remains constant and can be represented by a fixed delay T^c. An example of the timing of the actions of the control system is shown in figure 3.2. We see that the output of the system becomes, due to these random variations, time-varying.

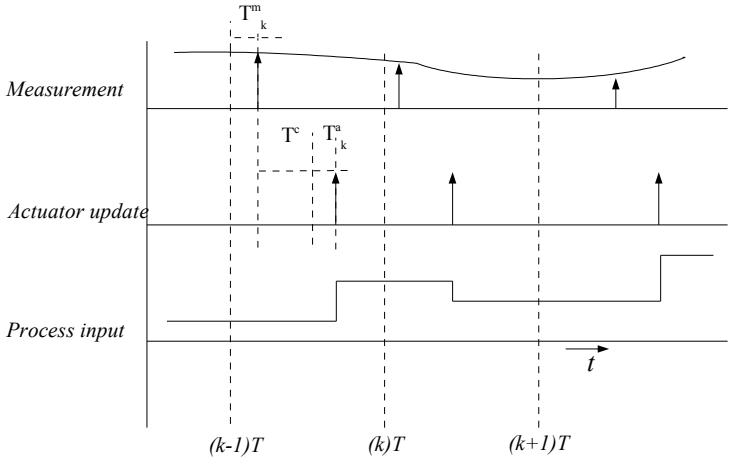

Figure 3.2 Timing of the signals in the control system. The first diagram shows the measurement that is delayed, the second diagram shows the computation and actuator delay and the third diagram shows the resulting process input.

3.2. TYPES OF DELAYED ACTIONS

We consider actions that occur in time and define different types based on their behaviour and properties. The first type we define is the synchronous actions:

Definition 3.1 Synchronous actions
Actions occur synchronously in time if the period length between each two successive actions is the same.

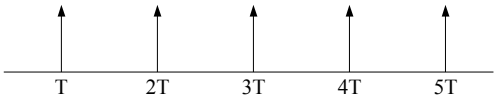

Figure 3.3 Synchronous actions. Defined by a period time T.

These actions are equidistantly spaced in time and we are able to find a period time T that determines each action (see figure 3.3). Extending this type brings us to the semi-synchronous actions and is defined as follows:

Definition 3.2 Semi-synchronous actions
Actions occur semi-synchronously in time if there exists a partitioning of the time scale into periods with equal lengths, such that each period contains exactly one action.

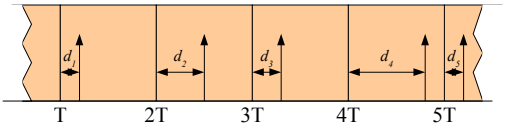

Figure 3.4 Semi-synchronous actions. Defined by a period time T and a delay d_k. The delay cannot become bigger than the period time T.

To identify these actions we have to define a period time T and a delay d_k that can change each period k. This is shown in figure 3.4. The delay d_k cannot become bigger than the period time T.

All discussed actions so far are all correlated with time. Asynchronous actions do not depend on some time variable and are defined as follows:

Definition 3.3 Asynchronous actions
Actions occur asynchronously in time if there does not exists a partitioning of the time scale into periods with equal lengths, so that each period contains exactly one action.

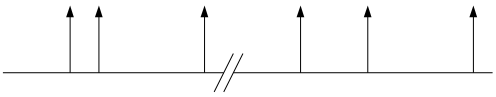

Figure 3.5 Asynchronous actions. Actions that are not correlated with some time variable. It can be possible to describe the actions stochastically.

It is very difficult to describe the occurrence of these type of actions. However, when there exists some probability interval in which an action takes place, we are able to describe the actions stochasticly way. Figure 3.5 shows some representation of asynchronous actions in time. At last we want to discuss a special type, the event-driven actions, that are defined by:

Definition 3.4 Event-driven actions
Actions occur event-driven if some event was the reason for this action to occur. This event can occur synchronously, semi-synchronously or asynchronously in time.

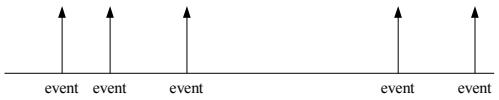

Figure 3.6 Event-driven actions. Actions that only occur based on an event. The actions can behave as synchronous, semi-synchronous and asynchronous, this depends on how the events are generated.

If an action can be related to some event in the system, we speak of event-driven actions. Only when this event becomes active, the action is initiated (see figure 3.6). We can describe this as some "super" type of the three discussed types, because the event-driven actions can behave like all other three (see figure 3.7). For example, if the events occur synchronously in time the actions are initiated synchronously in time too. By changing the timing of the events, we control the behaviour of the actions. We can generate events based on, for example, the error of the control system. In this case the events are not correlated with time, but a particular system variable. The resulting actions that occur can become a-synchronously. A problem with a-synchronous actions is that they are difficult to describe mathematically. This is mainly due the fact that we cannot relate the function to a time variable.

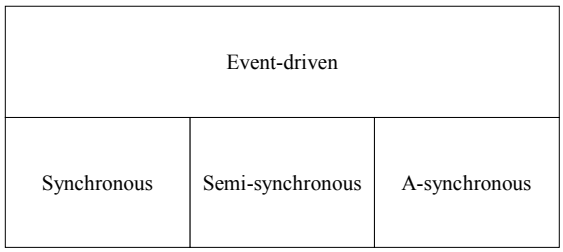

Figure 3.7 Class of the different action type. Event-driven is the super class, because the action can behave like all other three types.

4. INTRODUCTION TO CONTROLLER ANALYSIS VERSUS DELAYS

Fixed time delays are common in the control theory. These delays can complicate the controller design, but can be well described and that is also the reason that fixed time delays are often used in the controller design. Due to the fact the controller systems become more complex and the software engineers cannot guarantee strong timing requirements, we need to deal with variable delays in our analysis. The description of variable delays in controller systems is very complex and there is not much theory about this subject available. This chapter discusses how to describe fixed and possibly variable delays. Section 4.1 describes fixed time delays and in section 4.2 we discuss some topics of variable time delays.

4.1. FIXED DELAYS

Fixed time delays can be simply described in the system equations, because they can be represented with a constant value [2, 14]. When a controller system has a known varying delay, usually the worst-case time delay is taken as constant delay. The solution of the controller becomes more suitable for the new situation. This chapter discusses the known theory domains: state-space and the sampled-data domain.

4.1.1. STATE-SPACE

We consider in this section only the description of systems that are described in state-space. A fixed time delay can be described in state-space. We assume that the introduced delay τ is semi-synchronous, thus smaller than the sampling period T. The system is described by

$$\dot{x}(t)=Ax(t)+Bu(t-\tau). \tag{4.1}$$

The system is modelled by the zero-order-hold sampling model. Integration of the system over one sampling period gives

$$x(kT+T)=e^{AT}x(kT)+\int_{kT}^{kT+T} e^{A(kT+T-s')}Bu(s'-\tau)ds'. \tag{4.2}$$

Normally, we assume that the signal *u(t)* is piecewise constant in the sampling period. However, in this case the signal will change within the sampling interval, due to the introduced time delay. Therefore, we will split the integral of 4.2 into two parts, in which the signal *u(t)* will be constant again, and becomes

$$x(kT+T)=e^{AT}x(kT)+\int_{kT}^{kT+\tau} e^{A(kT+T-s')}ds'Bu(kT-T)+\int_{kT+\tau}^{kT+T} e^{A(kT+T-s')}ds'Bu(kT). \tag{4.3}$$

Rewrite the equation to the normal form gives us

$$x(kT+T)=\Phi x(kT)+\Gamma_1 u(kT)+\Gamma_2 u(kT-T), \tag{4.4}$$

where

$$\Phi=e^{AT}$$

$$\Gamma_1=\int_0^{T-\tau} e^{As}ds\,B$$

$$\Gamma_2=e^{A(T-\tau)}\int_0^{\tau} e^{As}ds\,B$$

The time delay introduces an extra controller value, which represents the past value of the controller. When we want to describe this system in a state-space model, we introduce this extra value as a state into the state-space equations. By taking $x_2(kT+T) = u(kT)$ we get the state-space of 4.4 and is given by

$$\begin{bmatrix} x_1(kT+T) \\ x_2(kT+T) \end{bmatrix} = \begin{bmatrix} \Phi & \Gamma_2 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(kT) \\ x_2(kT) \end{bmatrix} + \begin{bmatrix} \Gamma_1 \\ I \end{bmatrix} u(kT) \tag{4.5}$$

In the discrete state-space we are able to add extra states in order to calculate the output signal with time delays. We can extend this to delays that are bigger than the sampling time. All the extra states that are needed to calculate the system output are taken into account [2]. We will not discuss delays that become bigger than the period time, because we will only consider semi-synchronous delays.

4.1.2. Sampled-data domain

Sampled-data systems contain both discrete and continuous time parts. The power of the frequency domain analysis is that both parts are considered at the same time. Continuous time systems are described with Laplace equations and discrete time systems with z-domain equations. Both equations can be transformed into their frequency domain equivalent. So, actually we have to find the description of time delays into these two domains. The theory of the Laplace transformation gives us an important time delay property and is given by

$$\mathcal{L}\{f(t-\tau)\} = F(s)e^{-s\tau}. \tag{4.6}$$

This exponential is widely used to represent computational delay, transport delay, etc. It is a nice property that can be easily used to describe fixed delays. However, we see that the phase of the system is influenced by this exponential and can make controller design a tough job. If we look to the z-transform theory we can describe the relation to the Laplace transform by

$$z^k = e^{ksT}. \tag{4.7}$$

We see that the exponential defines a delay that depends on the sampling instant k. Therefore, we are able to represent a time delay that is a multiple of the sampling period T.

4.2. Variable delays

We cannot simply replace a variable delay for a fixed worst-case delay in every controller design. There are examples where the closed loop system is stable for all constant delays, but give instability when the delay is varying. More often, embedded systems introduce variable delays into the controller calculations. As result that in some cases the control criteria, such as performance and stability, are not met. Unfortunately, not much theory is developed to analyse these variable delays.

However, interesting work is done in the design of discrete control over a network [19]. In this case several delays are introduced in the controller loop due to sending and retrieving information over a communication network. A controller node communicates with a sensor and an actuator node. The delays that are introduced are varying in a random fashion. Random delays which are independent every transfer and random delays with a probability distribution governed by an underlying Markov chain are studied. The control system is described, as discussed in section 4.1.1., as a discrete state-space model in both approaches. An LQG-design method is developed when the delays have a constant known density function.

Another approach is event-driven control [1]. Only when an event is initiated the control algorithm calculates a new actuator output. An event can be generated in many ways as discussed in chapter 3. For example an event could be generated when the error (or derivation of the error) of the

system becomes too big. A reason why event-based control is interesting is that it is closer to nature. For example, the human motion control is event triggered and not time triggered. Normally, a control algorithm calculates a new actuator value every sampling period. When the output does not change very much, the controller actually wastes computation time. Therefore, event-driven controller algorithms can be of special interest for embedded implementation on software platforms. However, this approach cannot be analysed with conventional methods.

5. GLOBAL SENSITIVITY ANALYSIS

Global sensitivity analysis[3] in this case is to determine the overall sensitivity of a particular sampled-data system to variations in the sampling frequency. In this chapter we take a very small and fixed delay to analyse the behaviour of the system. Most analysis of sampled-data systems is done in the frequency domain, because both of the discrete and the continuous time parts of the system can be considered at the same time. Based on the ideas in paper [3] we develop a sampling sensitivity function in the frequency domain that represents the sensitivity to a small sampling period variation ΔT. Actually we consider a sampling period that slightly differs from the ideal sampling period, see figure 5.1. Often, embedded systems are implemented with a low costs internal oscillator and that results often in an inaccurate clock frequency for the microprocessor. This is an example in which the sampling frequency will be different than the frequency for which the controller originally was designed.

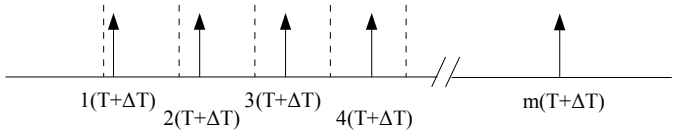

Figure 5.1 The sampling period error becomes every instant bigger with ΔT.

Section 5.1 describes how the response of a sampled-data system can be described in the frequency domain. Also two important transfer functions are discussed which are used in further sampling sensitivity analysis. The theory of the sampling sensitivity function is described in section 5.2. Two controller system examples are analysed with the sampling sensitivity function and is described in section 5.3. Finally, we evaluate the sampling sensitivity function in chapter 5.4.

5.1. FREQUENCY RESPONSE OF SAMPLED-DATA SYSTEMS

In sampled-data systems both discrete and continuous time signals are considered at the same time. All the signals in a sampled-data system can be transformed to their equivalent in the frequency domain. Therefore, we want to analyse the system in this domain. In this section we shall briefly discuss how we get the response and transfer function of a given sampled-data system in the frequency domain. Appendix A shows the derivation of the response in more steps. We introduce the impulse modulation model to describe sampled signals [2].

If we define a continuous time signal *r(t)*, its sampled version *r*(t)* and the period time *T* we can give the relation of the sampled signal by

$$r^*(t) = \sum_{k=-\infty}^{\infty} r(t)\,\delta(t-kT). \tag{5.1}$$

With this method the sampled signal is described in the continuous time domain and it is allowed to transform the signal to its Laplace equivalent by

$$R^*(s) = \mathscr{L}\{r^*(t)\} = \int_{-\infty}^{\infty} r^*(t)\,e^{-sT}\,dt. \tag{5.2}$$

Substitution of equation 5.1 into 5.2 and replacing the delta function with its fourier series representation gives us

3 Local analysis is when the system is analysed between the sampling instants and is done mostly in the time domain.

$$R^*(s)=\int_{-\infty}^{\infty} r(t)\left\{\frac{1}{T}\sum_{n=-\infty}^{\infty} e^{jnw_s t}\right\}e^{-sT}\,dt. \tag{5.3}$$

With $w_s=2\pi/T$. Changing the integral and the summation and using the properties of the Fourier transformation we get

$$R^*(s)=\frac{1}{T}\sum_{n=-\infty}^{\infty} R(s-jnw_s) \tag{5.4}$$

Where *R(s)* is the Laplace transform of *r(t)*. Equation 5.4 shows that a sampled signal contains infinitely many copies of the spectrum of the original continuous time signal. The phenomenon of aliasing is the overlap of the spectral copies (frequency folding) of the sampled signal. With sampled-data systems we have always to make sure that we avoid aliasing. Mostly, before sampling, a filter is placed that removes all spectral contents above half the sampling frequency. Then there will be no overlap and thus no aliasing.

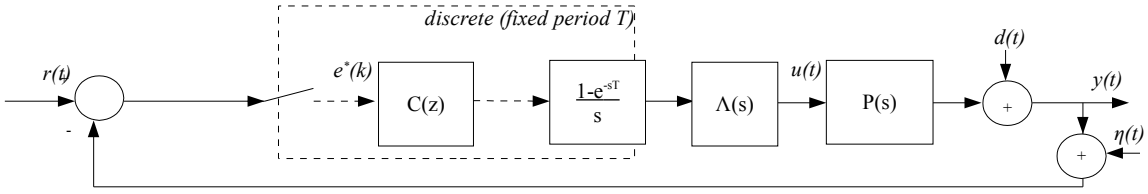

Figure 5.2 Scheme of a sampled-data system.

We consider linear time-invariant systems in the continuous and discrete time and they both can be described by a transfer function. A discrete transfer function is described in the z-domain and a continuous transfer function in the s-domain (Laplace). Given a continuously time transfer function such as *P*(s)*, we can find the corresponding z-transform by letting $e^{sT} = z$ or

$$P(z)=P^*(s)|_{z=e^{sT}} \tag{5.5}$$

This is an important property that is used to determine the response or the transfer function of a sampled-data system. Figure 5.2 shows the block scheme of a standard sampled-data system implementation. Block *Λ(s)* introduces a fixed computation delay to the system. We describe the output *Y*(s)* by

$$\begin{aligned} Y^*(s)=&\frac{(1-e^{-sT})C^*(s)\left(\frac{\Lambda(s)P(s)}{s}\right)^*}{1+(1-e^{-sT})C^*(s)\left(\frac{\Lambda(s)P(s)}{s}\right)^*}(R^*(s)-N^*(s)) \\ &+\frac{1}{1+(1-e^{-sT})C^*(s)\left(\frac{\Lambda(s)P(s)}{s}\right)^*}D^*(s). \end{aligned} \tag{5.6}$$

We are able to obtain the sampled-data sensitivity and the complementary sensitivity function in the same way as we know from the continuous time theory. Using equations 5.4, 5.5 and 5.6 we can define the sensitivity function as

$$S^*(s)=\frac{Y^*(s)}{D^*(s)}=\frac{1}{1+(1-e^{-sT})C(e^{sT})\frac{1}{T}\sum_{n=-\infty}^{\infty}\left(\frac{\Lambda(s-jnw_s)P(s-jnw_s)}{s}\right)}, \tag{5.7}$$

and the complementary as

$$T^*(s)=\frac{Y^*(s)}{R^*(s)}=\frac{(1-e^{-sT})C(e^{sT})\frac{1}{T}\sum_{n=-\infty}^{\infty}\left(\frac{\Lambda(s-jnw_s)P(s-jnw_s)}{s}\right)}{1+\left\{(1-e^{-sT})C(e^{sT})\frac{1}{T}\sum_{n=-\infty}^{\infty}\left(\frac{\Lambda(s-jnw_s)P(s-jnw_s)}{s}\right)\right\}}. \tag{5.8}$$

When we take *n=0* we only consider the fundamental harmonic of the transfer function. Adding more harmonics to the function we get more spectral information. Because the base spectra contains the most information, we can simplify the equations 5.7 and 5.8 and only consider the fundamental harmonic. Therefore, we define the fundamental sensitivity function by

$$S_f(s)=\frac{1}{1+\frac{1}{T}\frac{(1-e^{-sT})}{s}C(e^{sT})\Lambda(s)P(s)}, \tag{5.9}$$

and the fundamental complementary sensitivity function by

$$T_f(s)=\frac{\frac{1}{T}\frac{(1-e^{-sT})}{s}C(e^{sT})\Lambda(s)P(s)}{1+\left\{\frac{1}{T}\frac{(1-e^{-sT})}{s}C(e^{sT})\Lambda(s)P(s)\right\}}. \tag{5.10}$$

5.2. GLOBAL SAMPLING SENSITIVITY FUNCTION

We define a function that represents the sensitivity to small sampling period variations of a particular transfer function of a sampled-data system. In principle, every transfer function is allowed. However, we will only consider the transfer functions that are given by equation 5.9 and 5.10 in our analysis. They reflect the most important properties of the controller system, such as tracking and disturbance rejection.

The z-transform is defined for a fixed period T, therefore it is not allowed to vary T. However, if we consider only significant small variations of the period time, then the definition of the z-transform will hold in the analysis. To calculate the sensitivity to small variations of a particular transfer function, we define the sampling period as $T = T_f + \Delta T$. The sampling period consist of a fixed period and a small period error; $\Delta T << T_f$. Actually, the algorithm is executed with a slightly different period time as determined in the controller design.

In order to get the sensitivity to the period time T, we differentiate the given transfer function to the period time. Thus, the transfer function must be a function of T also[4]. The definition of the differentiation shows that the variable ΔT is indeed significantly small. We define the sampling sensitivity function by

$$SSF_H(s)=\lim_{\Delta T\to 0}\frac{H(s,T+\Delta T)-H(s,T)}{\Delta T}=\frac{dH(s,T)}{dT} \tag{5.11}$$

The result of $SSF_H(s)$ is the derivation, to the sampling period, of transfer function H and represents

4 If the transfer function is not a function of the sampling period, then it will be robust to sampling period variations.

the sensitivity to small sampling variations of this transfer function. In other words, if the transfer as function of T is very steep, it will change a lot if the sampling period slightly differs from the ideal one. With the *SSF* function we can determine how big this difference will be.

The sensitivity functions of equations 5.9 and 5.10 will be used in further analysis, because the sensitivity and the complementary sensitivity represent important properties of the controller system [19]. Therefore, we shall describe the SSF function of these particular transfer functions. A known property of the sensitivity functions is $S + T = 1$. We extend this to the SSF function domain and we see that it becomes $SSF_S + SSF_T = 0$. This means that the gains of the SSF functions of these two sensitivity functions are exactly the same. However, we notice that the the phase is shifted with 180 degrees. The advantage is that we only have to calculate one particular SSF function in order to get the sensitivity of the controller system that contains all aspects that is represented by the two sensitivity functions. We shall develop the SSF_S based on equation 5.9 and gives us

$$SSF_S(s) = -SSF_T(s) = \frac{\frac{1}{T}\Lambda(s)P(s)}{1+\left(\frac{1}{T}\Lambda(s)P(s)ZOH(e^{sT})C(e^{sT})\right)^2}\frac{d\left(ZOH(e^{sT})C(e^{sT})\right)}{dT} \tag{5.12}$$

5.3. SSF ANALYSIS

We consider the control scheme as shown in figure 5.2. The computation time of the controller algorithm is neglected. For the plant and the controller we take

$$P_1(s) = \frac{1}{(s+5)}, \quad \text{and} \quad C_1(z) = \frac{25T}{8}\frac{z+1}{z-1}.$$

This is a stable first order plant. The controller is designed in the continuous time domain and transformed to its discrete time equivalent with the Tustin transformation. In practice such designs

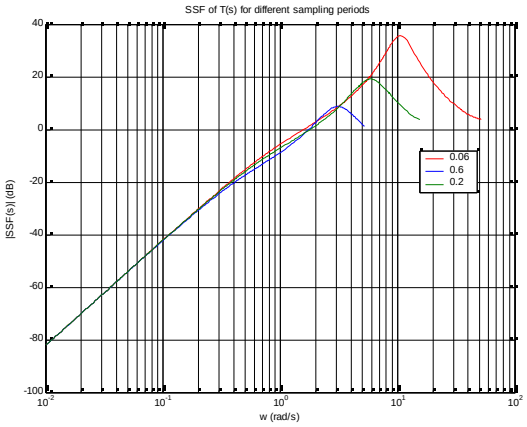

Figure 5.3 $SSF_T(s)$ for different sampling periods.

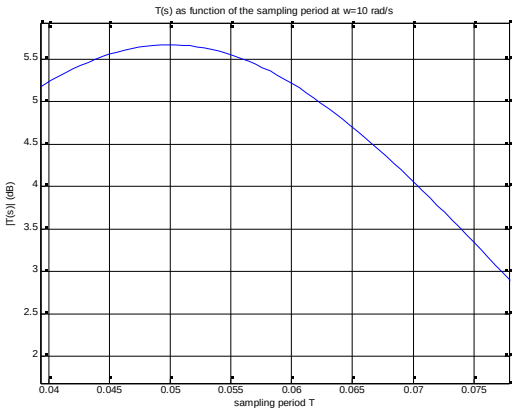

Figure 5.4 T(s) as function of the sampling period T at w=10 rad/s.

are the order of the day, therefore we chose this approach. As rule of thumb, we take twenty times the system bandwidth as first selection of the sampling frequency. In this case we start with $T = 0.06$. We compute the *SSF(s)* of the complementary sensitivity transfer function *T(s)*; equation 5.10. Figure 5.3 shows the result for different sampling periods and the function is plotted on a logarithmic scale. The part of the function that is below zero shows very small values and tells us that the transfer function is not sensitive to sampling variations. On the other hand, all that is above zero shows bigger values and we have to conclude that the transfer will be sensitive to sampling

period variations. If we look to the graph we see that the line is very straight till the peak. We can approximate this with a linear tangent. The slope of this line shows the relation between the sensitivity to sampling variations and the excited input signal to the system. We see that the system becomes more sensitive at higher frequencies. Very trivial, because it is clear that a sampled-data

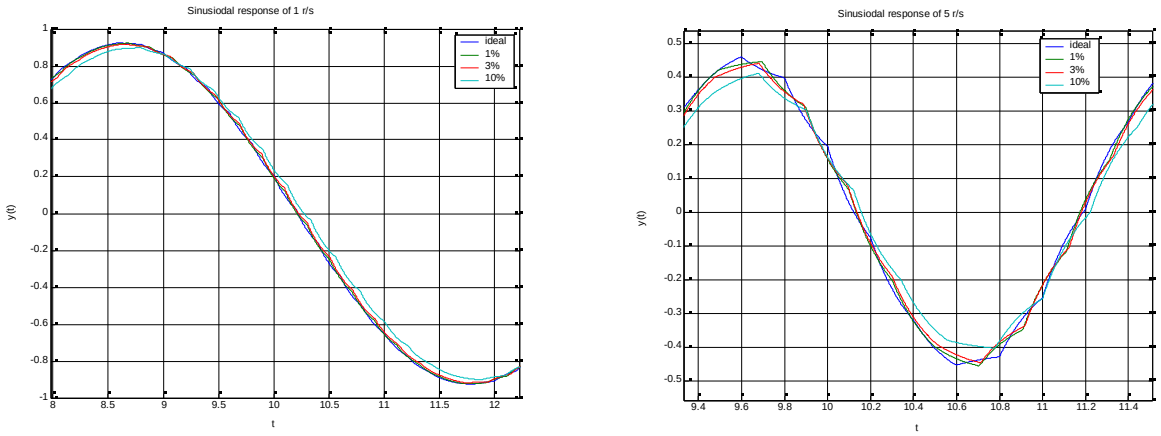

Figure 5.5 Response of the control system with a sine as reference. The system at the left is excited with 1 rad/s and at the right with 5 rad/s.

system has more difficulties to follow signals with a high frequency than with a low frequency. But it is a nice observation that this line represents this relation.

We see in figure 5.3 that the SSF function has a maximum at 10 rad/s. This means that the transfer function at this point has a steepness of 35.41dB. Figure 5.4 shows the complementary function *T(s)* as function of sampling period *T* at frequency 10 rad/s. So, if the sampling period differ with 1%, we see that the transfer function actually drops with approximately 0.02dB. At first sight, we could think that the system is very sensitive in this peak, but this difference is not significant to influence the behaviour of the transfer function. In fact, the $SSF_T(s)$ shows us that the transfer function is not very sensitive at all. Over the total range the values are very small when the sampling period differs with 1%. However, this observation is difficult to see in the graph of figure 5.3. We will normalize the *SSF(s)*, so that we immediately see the result of the sensitivity at each point when the sampling period is changed with n%. In our case *n* becomes one. Therefore we define a normalized equation of 5.11 and is given by

$$SSF_H^{n\%}(s) = \frac{H(s,T)}{dT} \cdot \frac{nT}{100} \tag{5.13}$$

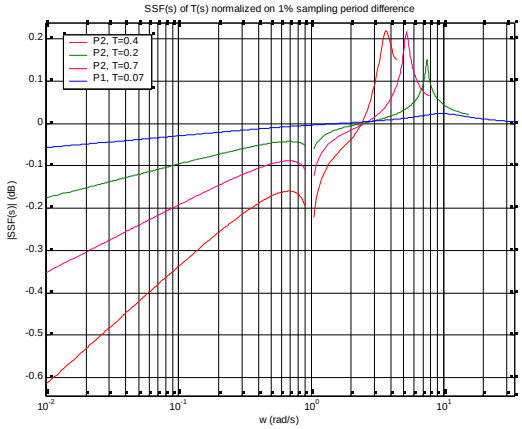

Figure 5.6 $SSF_T(s)$ for different sampling periods.

This function represent the sensitivity to sampling period variations when the sampling period is changed with n%. The blue line of figure 5.6 shows the result of the normalized function to 1% of the controller system based on a sampling period T = 0.06.

If we take a look at the function with sampling period T = 0.2 sec in figure 5.3, we see a peak at 5 rad/s. Because the *SSF(s)* shows a difference between the frequencies 1 and 5 rad/s, so we are able to see the influence of sampling variations in the response of the system also in these cases. In order to do this, we excite the system sequentially with a sinusoid of frequency 1 and 5 rad/s. Figure 5.5 shows the response of the system with different sampling period errors. We notice that the system response at the right differs much more than at the left picture when the sampling period is slightly changed. Thus, the system is indeed a bit more sensitive when it is excited to higher frequency signals. However, the difference between the ideal and the 1% period variation is not too big.

Lets consider an "unstable" process (oscillator) and a controller that stabilizes the plant. The influence of sampling variations in this case is assumed to be bigger than with the previous system. We use the following plant and controller

$$P_2(s)=\frac{1}{(s^2+1)}, \quad \text{and}$$

$$C_2(z)=\frac{25T}{4}\frac{(z+1)(16z^2-32z+16+8Tz^2-8T+5T^2z^2+10T^2z+5T^2)}{(z-1)(5Tz+5T+z-1)^2}.$$

Figure 5.6 shows the normalized $SSF^{1\%}{}_T(s)$ of controller system and also of the previous system, so we can see the difference between the controller examples. The sampling frequency is chosen at 0.2 sec. We see that the control system of the unstable plant is indeed more sensitive to sampling variations than the previous system; a factor 6 more. Because, there is a lot more difference between the frequencies 1 and 5 rad/s as with the previous stable system, we excite this system also with a sine function at these two frequencies. The response is shown in figure 5.7. At the left picture the ideal response is very similar as the one with 1% sampling period difference. This is not the case at the right; here we can see that the response is different as the ideal one. Therefore, we notice that the result is consistent with the result of the *SSF(s)* function. However, the peak, of P2 with T = 0.2, in figure 5.6 shows that the transfer function *T(s)* is amplified with 0.15 dB and this difference is very small. So, we cannot speak of a significant different behaviour of the transfer function when it is exposed to a slightly different sampling frequency.

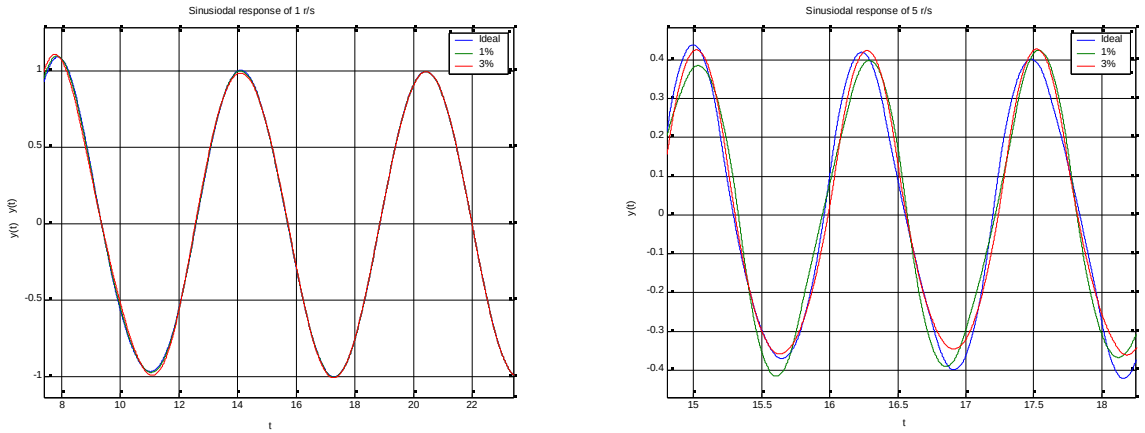

Figure 5.7 Response of the control system with a sine as reference. The system at the left is excited with 1 rad/s and at the right with 5 rad/s.

5.4. EVALUATION

We saw that the response and the transfer of a sampled-data system can be described in the frequency domain. These equations have to be based on the sampling period T as well. Because the Matlab control toolbox cannot calculate with transfer functions that contain a symbolic sampling period variable. Therefore, we have written our own functions for this analysis method. Appendix B describes these m-files and shows how to calculate the sampling sensitivity function.

With the sampling sensitivity function (SSF) we are able to determine the sensitivity to sampling period variations of a given transfer function. Using the normalized equation of the SSF we can immediately display the influence that a sampling period, with an error of n%, has on the transfer function. The steeper the line of the SSF function, the more sensitive the controller system will be to sampling period variations.

The small change in the sampling period introduces also a very small change in the transfer function. This means that the behaviour of a system will not change significantly. However, the SSF function shows a tangent (till the peak) which is a linear line that represents that the system becomes more sensitive to sampling variations when it has to deal with higher input frequencies. The SSF function gives some insight in sampled-data systems that are exposed to a slightly different sampling period. However, the SSF function shows global sampling period sensitivity information of a particular system. Unfortunately, it cannot be used to analyse variable sampling delays that are not significantly small compared with the sampling period. In this case the definition of the z-transform does not hold. For the analysis of variable sampling delays we have to look further to other methods.

6. MODELLING VARIABLE DELAYS

Sampled-data systems contain both discrete and continuous time signals. Due to the sampling-and-hold elements that are introduced to the system, it becomes time-variant [2]. When the controlled system is exposed to varying sampling period and actuator update delay variations, the total system becomes nonlinear. The mathematical description of these kind of controller systems with delay variations is not defined yet. All discrete controller and sampled-data theory is based on equidistant sampling and cannot be applied in this situation.

We want to describe controlled systems, that are exposed to sampling and calculation delay variations, with linear and time-invariant equations. A general idea to solve this, is to extract all time-varying and nonlinear parts of the system, that is caused by the introduced delay variations. Then we are able to describe the control systems with linear and time-invariant equations in the z-domain for discrete or the s-domain for continuous time. By introducing "special" disturbances the time-varying and the non-linear behaviour can be fed back into the system. In this case we have divided the complex varying sampled-data system into two parts, namely a linear time-invariant that can be mathematically described and a time-variant and nonlinear part that is represented by the disturbances. Analysis of the complex behaviour of the delay variations is then based on the content of the disturbance signals. Thus, all time-varying and non-linearity is captured in these special introduced disturbances.

We will describe in section 6.1 each delay separately and show how to represent the delay variation by external disturbances. Section 6.2 we describe the total model in which all the delays and disturbances are introduced. The disturbance equations can be transformed into two loops in the controller scheme. In section 6.3 we discuss the transfer function of the total model.

6.1. DISTURBANCES TO REPRESENT DELAY VARIATIONS

We consider only measurement, computation and actuator update delays, as discussed in chapter 3. The measurement and actuator update delay vary in time and the calculation delay is assumed to be constant. This section describes how we extract these delay variations of the system and bring them back into the system in the form of disturbance inputs. Because we have two different variable delays, measurement and actuator update, we describe them separately in two sections.

6.1.1. VARIABLE MEASUREMENT DELAYS

For understanding the variable measurement (or sampling) delays, we consider the sampled error signal more closely. What is the big difference between the real situation and that we use in our theory. Figure 6.1 shows the samples of the signal at the ideal and the delayed sample times. We

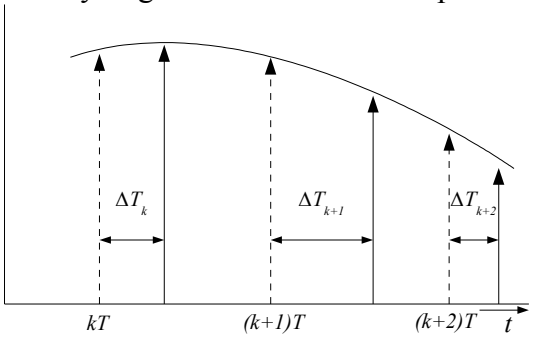

Figure 6.1 The delayed samples (solid) are the sample values read by the controller in the practice. Analysis is based on the ideal (dashed) samples.

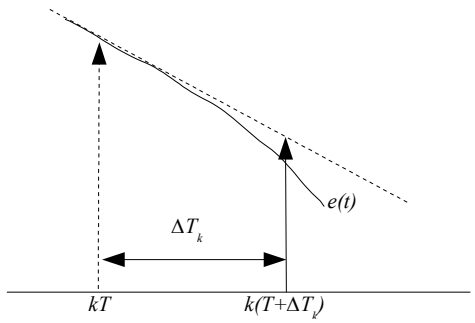

Figure 6.2 By extrapolating the ideal (dashed) sampled value, based on the delay, the delayed sampled value (solid) can be approximated.

consider only semi-synchronous delays, so the delay ΔT_k can be different every instance k but cannot become greater than the period time T. Note that the controller algorithm always assumes, in its calculations, that the samples are equidistant in time. Also equidistant sampling is considered in the existing controller theory. In practice this is not the case. The controller will calculate with other values, namely the delayed ones, so we have to modify the standard analysis method. Analysis of the controller is done in a discrete time way, at time instance k, and is by definition synchronous. Therefore, shifting the samples in time brings us back to the difficulty that we wanted to avoid in the first place.

To bring the sampling delay into the system description we have to do two different steps: adapting the sampled value at the input of the controller and delay the controller output before it goes into the plant. Adapting the ideal sampled value to the value that will be actually measured in practice. Figure 6.2 shows that in the analysis the controller uses the sample at the ideal sample moment kT, the dashed arrow. Assuming that the measurement delay is known for each instant k,

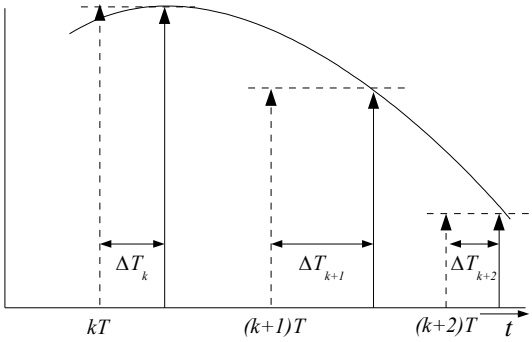

Figure 6.3 Shifting back of the delayed samples (solid) to the ideal sample moments (dashed).

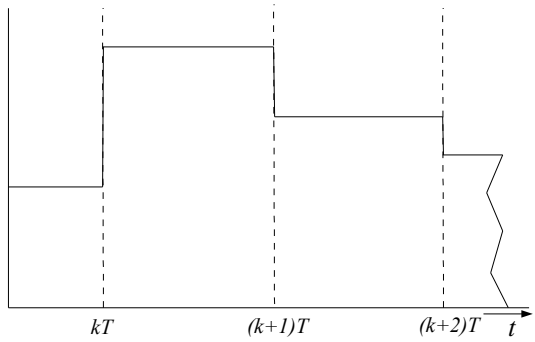

Figure 6.4 ZOH output without delay.

we can approximate the sampled value that will be read by the controller in the real situation. This can be done by extrapolating the error signal. Figure 6.2 shows a linear approximation of the error function as the first consideration. In other words, we actually shift all the delayed samples back to the ideal sample moments in the analysis (see figure 6.3), so that we are still able to describe the discrete signal on each sampling instant. We introduce disturbance input $d_e(t)$ with which we adapt the sampled values (see figure 6.6).

We have adapted the ideal sampled values of the analysis to the sampled values that will occur in real situation. However, the measurement delay ΔT_k that is introduced is not compensated yet. Therefore, we have to bring the delay ΔT_k into the system. The controller uses samples that actually occur ΔT_k seconds in the future. This imply that the controller updates the actuator value ΔT_k seconds too early. If we do not take this delay into account, the plant is excited with a wrong

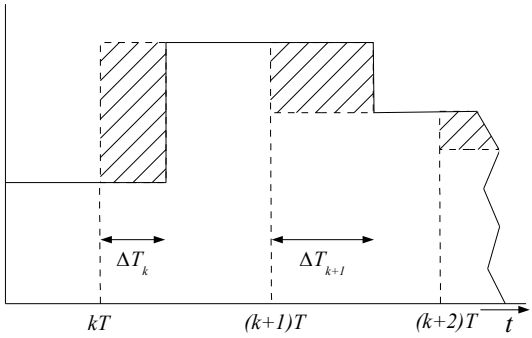

Figure 6.5 Adding a blockform to the ZOH output results in a delayed output signal, which is equal to a delayed controller output.

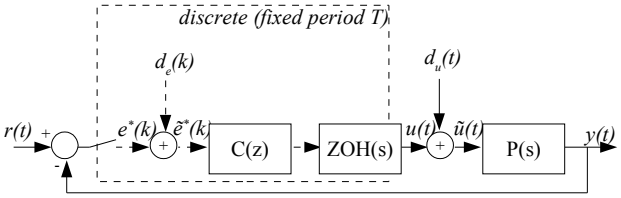

Figure 6.6 Controller structure with the introduced disturbance inputs to expose the system to measurement delay variations. The sampler, controller and ZOH are described on fixed sampling period with period time T.

input signal (see figure 6.4). To compensate for this delay, we have to delay the output of the controller before it goes into the plant. We are able to delay the controller output after the *ZOH*, because the signals at that point are continuous time signals. It is easy to see that we have to add different step functions to the *ZOH* signal in order to delay the controller output, see figure 6.5. Each instant k we have to adapt the *ZOH* output, so that it is delayed for ΔT_k seconds. This is done by a special introduced disturbance input $d_u(t)$ (see figure 6.6) that adds a stepwise continuous function, which is described later in this section, to the output signal of the *ZOH*. We call the introduced disturbances "special", because they are not similar to the disturbances we know of the controller theory. Normally a white noise is fed into the system via these inputs, but these disturbances are signals that are dependent of the state of the controller system.

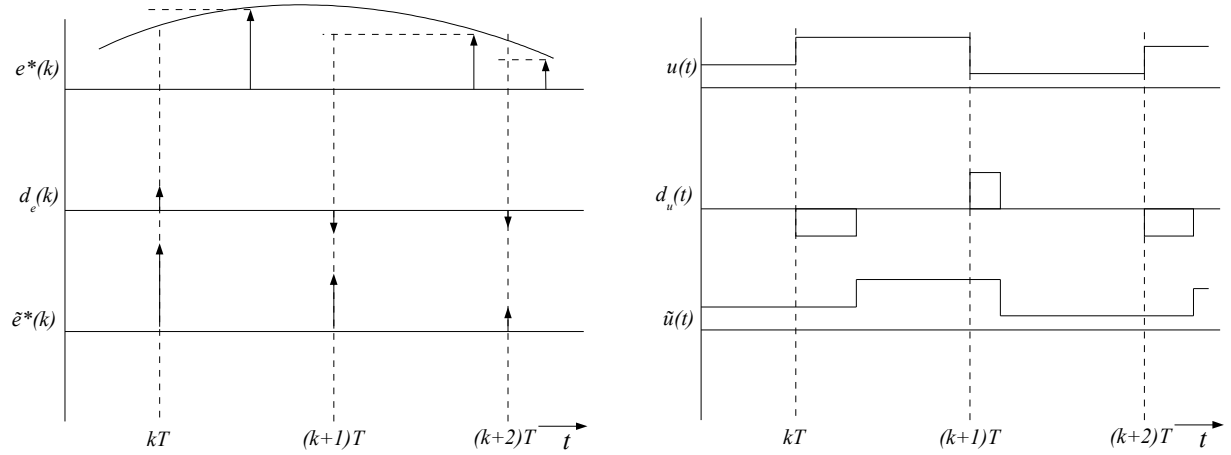

Figure 6.7 (a) shows the signals that are involved to adapt the sampled error signal. Signal ẽ(k) describes the delayed samples that are approximated by extrapolation. The disturbance value is the difference between the value at time kT and its delayed value. This value is added to the sample value of e*(kT) and results in a new sample value that is processed by the controller. (b) shows all the signals before they are fed to the plant. The disturbances adapt the signals such that the output signal contains the delay variations. In this way the plant is excited with a delayed controller output signal.*

We have described how we get the measurement delay into the system, without complicating the system description, by introducing two special disturbance signals. One disturbance that adapt the sample values that goes into the controller and one disturbance that delays the *ZOH* output signal before it goes into the plant. We can note that these special disturbance signals are actually the error between the ideal and the real situation. These "error" signals are added to the ideal signals and turn the system into a system which is exposed to measurement delay variations. Figure 6.6 shows the model with the introduced special disturbance inputs. The disturbance input d_e is a discrete time signal and adds every instant k a value to the ideal sample, so that the sample actually gets the value of the delayed measurement that occurs in practice. We can describe this by

$$d_e(k) = \dot{e}(kT)\Delta T_k + \frac{\ddot{e}(kT)\Delta T_k^2}{2} + \frac{\dddot{e}(kT)\Delta T_k^3}{3} + \textit{higher order terms.} \tag{6.1}$$

Note that function $\dot{e}(kT)$ is the sampled value of the continuous function $\dot{e}(t)$, which is the derivative of the error $e(t)$. We neglect the higher order terms of the expansion of function $\dot{e}(t)$ and use the first order only to do a linear extrapolation to approximate of the delayed sampled value (see figure 6.2). The equation becomes then

$$d_e(k) \approx \dot{e}(kT)\Delta T_k. \tag{6.2}$$

This equation corrects the error signal that is going into the controller. Note that we can only use equation 6.2 when *e(t)* is defined as a smooth function. Because *e(t)* is composed of reference *r(t)* and output *y(t)*, these functions needs to be smooth as well. We approximate the delayed sampled value based on the delay ΔT_k. The disturbance that delays the output of the *ZOH*, to compensate the measurement delay, has to add a signal such that the plant is excited with a delayed controller signal. In this case the signal is a stepwise continuous time function. The disturbance d_u is described by

$$d_u(t) = \sum_{k=0}^{\infty} (u(k-1) - u(k))(\varepsilon(t - kT) - \varepsilon(t - kT - \Delta T_k)). \tag{6.3}$$

With *ε(t)* the unit step function. Figure 6.7 shows how the signals are adapted that are concerned with the introduced disturbance inputs, based on the controller system of figure 6.6. The error signal is sampled on ideal moments in time and is changed each instance to the value that actually will be sampled in practice. We see that the disturbance $d_e(k)$ is the difference between the sampled values of the ideal and delayed moments. The controller calculates with the adapted values of signal $\tilde{e}^*(k)$. All signals in this section are discrete and are described on each sampling instance. Signal *u(t)* represents the *ZOH* output of the controller on the ideal sample moments. Compensating the introduced measurement delay, we have to delay the continuous time *ZOH* signal by adding disturbance signal $d_u(t)$. The plant is excited with the adapted signal $\tilde{u}(t)$ and we see that this is the delayed version of *u(t)*. We notice that the introduced disturbance inputs represent the error between the ideal and real situation of the controller system. The advantage of this type of model is that we can deal with measurement delay variations together with the existing theory to describe a discrete time controller in combination with a continuous time plant.

6.1.2. VARIABLE COMPUTATION DELAYS

Besides the measurement delay variations, we have to deal with calculation delay variations of the executed algorithm. These delay variations are caused by the software platform, because for example different types of tasks running on the same processor. Due to scheduling, the beginning of the controller task can vary in time. Normally, the duration of a controller task is very constant. Thus, we see that the controller execution varies each period, see figure 6.8. This means that the actuator update of the controller shall not be equidistant in time and we see that the delays vary each sampling instant.

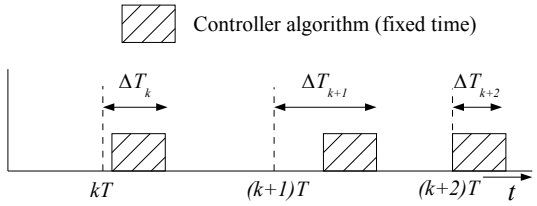

Figure 6.8 Due to the software platform, the calculation time of the control algorithm varies in time. However, the algorithm itself often has quite a constant calculation time.

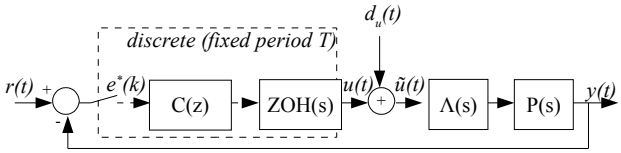

Figure 6.9 Controller system with an extra disturbance to bring the actuator delay variations back into the system. Block Λ(s) introduces a fixed computation delay. The sampler, controller and ZOH are described on fixed sampling period with period time T.

We noticed earlier that the calculation time of a controller algorithm on a processor is very constant. That is also the reason that in many analysis methods a fixed delay represents the worst-case calculation time of an algorithm. When dealing with calculation delays we can divide them into a constant and a time-varying part. The constant delay can be represented by a simple delay in the

controller scheme. However, when dealing with time-varying delays, we have many difficulties to describe such systems. That is the reason that we extract it from the system, so that the system can be described again with time-invariant equations. Actually, the plant is excited with a varying delayed controller output. To bring back the delay variations into the system we have to delay the controller output. This can be done only after the *ZOH* output, because at that point the signals are continuous. Therefore, we introduce an extra disturbance input after the *ZOH* that takes the variable delay of the controller output signal into account. Figure 6.9 shows the controller scheme with the fixed delay and the extra disturbance input. Figure 6.7b shows the signals also for this case, because it shows how the controller output is delayed. We see here how the disturbance input d_u compensates the output signal. The plant is excited with a controller output with a varying delay. Note that only semi-synchronous delays are considered and therefore the fixed and the varying calculation delay together cannot become bigger than the period time T.

Now we have obtained the correction term of the introduced disturbance input. This input has to add some signal to the controller output such that the output is delayed. We can describe the disturbance d_u by

$$d_u(t) = \sum_{k=0}^{\infty} (u(k-1) - u(k))(\varepsilon(t - kT) - \varepsilon(t - kT - \Delta T_k^a)). \tag{6.4}$$

It is a continuous stepwise time signal. The actuator update delay variations ΔT^u_k represents only the time-varying part of the calculation delay. The fixed delay can be described by existing theory, from the sampled-data domain.

6.2. MODEL WITH BOTH VARIABLE MEASUREMENT AND CALCULATION DELAYS

We know how we get the measurement and actuator update delay variations into a control system, without complicating the system equations. We bring both of variable delays into one model and describe the introduced disturbance signals. In the previous section the disturbances that bring these delays into the system are described. Now all the different disturbance inputs are combined. We introduce a standard sampled-data system, shown in figure 6.10. Before the error signal is sampled a filter is added to the system and eliminates the aliasing effect that is introduced when the signal is sampled [2]. Disturbance input d_e adapts the error samples before they are used by the controller algorithm. The delay variations of the measurement and the computation is represented by disturbance input d_u. Before the plant input the block Λ delays the controller output, representing the fixed computational delay.

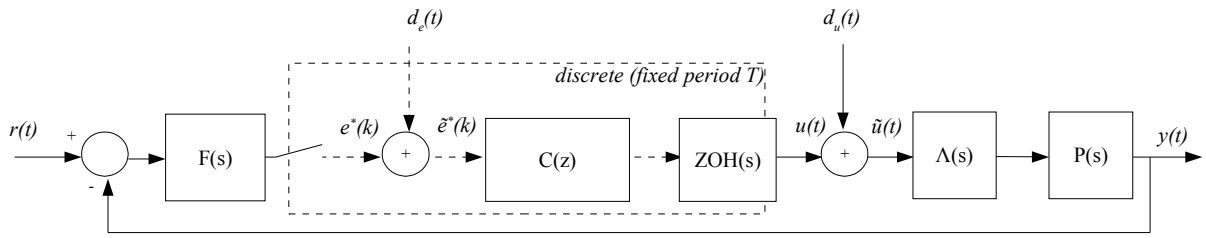

Figure 6.10 Model that contains both disturbance inputs to expose the system to sampling and calculation delay variations. Block Λ represents the fixed computation delay of the executed controller algorithm and F(s) represents the

The disturbance inputs reflect two different delay variations, namely the measurement and the actuator update delay. Measurement delay variations are given by ΔT^m_k and actuator update delays are given by ΔT^u_k for each instance k. We will describe these disturbances in the continuous time domain. For input d_e we use equation 6.2 and the Dirac function to transform it to a continuous

time function, this gives

$$d_e(t)=\sum_{k=0}^{k=\infty} \dot{e}(kT)\Delta T_k^m \delta(t-kT). \tag{6.5}$$

This disturbance only adapts the signal based on the measurement delay for each instance *k*. For disturbance d_u the equations 6.3 and 6.4 are combined

$$d_u(t)=\sum_{k=0}^{\infty} (u(k-1)-u(k))[\varepsilon(t-kT)-\varepsilon(t-kT-\Delta T_k^m-\Delta T_k^a)]. \tag{6.6}$$

The equation is based on the measurement as well as on the actuator update delay variations. By substitution of $\Delta u(k) = u(k) - u(k\text{-}1)$ and $\Delta T_k = \Delta T^m{}_k + \Delta T^a{}_k$ we obtain equation 6.7. Note that the delay is restricted to $\Delta T_k + T^v < T$, because we only consider semi-synchronous delays.

$$d_u(t)=-\sum_{k=0}^{\infty} \Delta u(k)[\varepsilon(t-kT)-\varepsilon(t-kT-\Delta T_k)]. \tag{6.7}$$

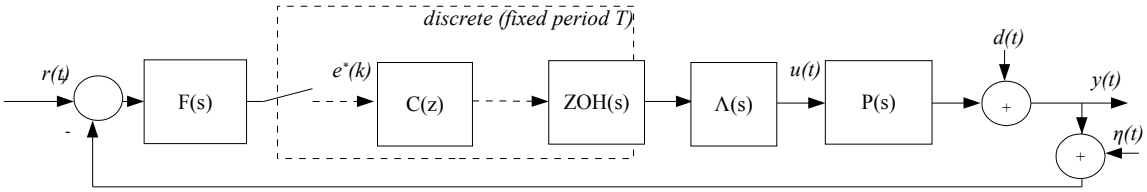

Figure 6.11 Controller structure with the standard two disturbance inputs. Block Λ represents the computation delay of the executed controller algoritmn.

Many controller design theories use two standard disturbances, namely actuator and sensor disturbances. Therefore, we transform the introduced special disturbances to these standard locations, such that we can use existing theory. The controller structure is shown in figure 6.11. Input *d* is the actuator disturbance and input η the sensor noise disturbance. We have to describe these inputs as function of our special introduced disturbances. The equation becomes convolution in the continuous time domain. First we consider disturbance input *d* and transform disturbance d_u. The equation becomes

$$d(t)=P\{\Lambda\{d_u(t)\}\}. \tag{6.8}$$

Disturbance input d_e can be transformed to disturbance input η and is described by

$$\eta(t)=F^{-1}\{d_e(t)\}. \tag{6.9}$$

Notice that filter *F* must be bi-proper, otherwise we cannot retrieve the inverse transfer function and analyse this disturbance signal. These equations are all convolutions and can be transformed to the frequency domain. We are then allowed to multiply each transfer block. So the equation d_u becomes

$$D(jw)=D_u(jw)\Lambda(jw)P(jw), \tag{6.10}$$

and d_e becomes

$$N(jw)=-\frac{D_e(jw)}{F(jw)}. \tag{6.11}$$

6.3. TOTAL TRANSFER OF THE MODEL

We have modelled the variable delays of an embedded controller system such that we can describe the system by linear and time-invariant equations. The equations of the disturbance signals, 6.5 and 6.7, depend on the states and signals of the controller system. This implies that there exist extra loops from the system to the disturbance inputs. When we want to analyse the transfer of the total system we have to take these loops into account. The transfer of the system becomes more complex, namely nonlinear and time-variant.

The two disturbance loops are described separately and we begin with $d_e(t)$. If we take equation 6.5, we see that the function depends on the continuous derivative of $e(t)$ and the variable measurement delay ΔT^m_k. The delta function transforms the continuous time signal into a discrete signal with period time T, so it can be added by the controller input. Figure 6.12 shows how the equation is represented by a block scheme. Disturbance input $d_u(t)$, equation 6.7, depends on the output of the controller at instance k and $k-1$. If we analyse the equation a bit closer, we see that the disturbance input actually depends on the discrete derivative of the controller output. Multiplying equation 6.7 by T/T gives us

$$d_u(t) = -T \sum_{k=0}^{\infty} \left[\frac{u(k)-u(k-1)}{T} \right] [\varepsilon(t-kT) - \varepsilon(t-kT-\Delta T_k)], \tag{6.12}$$

and the definition of the discrete differentiation appears into the equation. The difficult part of the equation is the step function, because this step function switches every instance k at another moment. For the convenience we define a function $ZOH'(\Delta T_k,s)$, which produces a block starting at kT with length ΔT_k. The scheme of this disturbance is shown in figure 6.13.

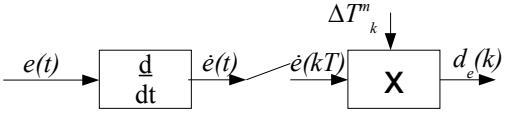

Figure 6.12 Scheme of the loop that is introduced by disturbance input $d_e(t)$.

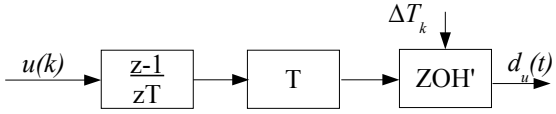

Figure 6.13 Scheme of the loop that is introduced by

It is remarkable that both disturbance equations actually depend on the time derivative of the signal that has to be modified. The analysis of chapter 5 gives us the same result when analysing the global sensitivity to sampling variations of discrete time systems. Basically, we write the discrete and continuous time system equations to its frequency response equivalent. For discrete systems this implies the substitution of $z=e^{st}$. When differentiating to the period time T, we get $z'=se^{st}$. This

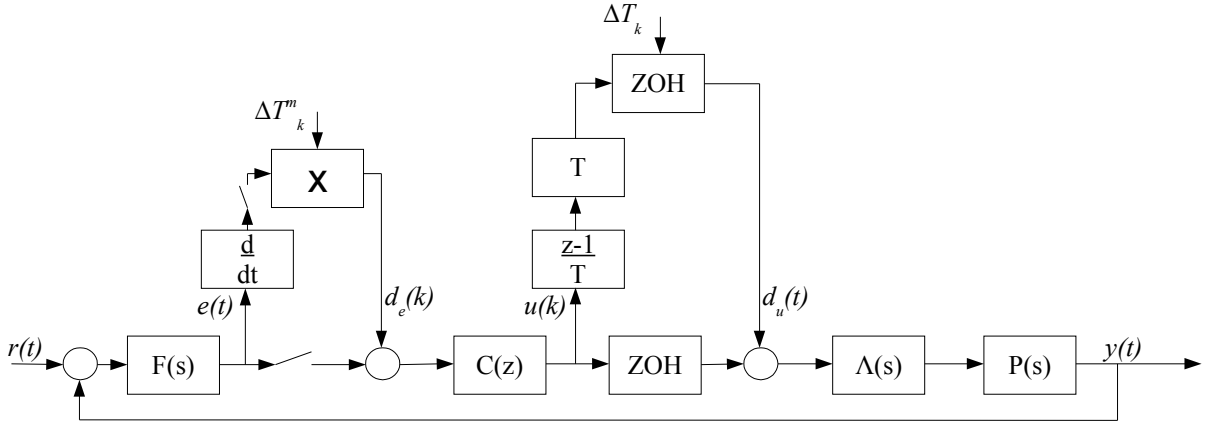

Figure 6.14 The total transfer of the model. Also the loops that are introduced by the disturbances are shown.

means that the sensitivity to sampling period variations depends on the derivative of the system signals. The model that is represented in this chapter, which describes the error between the real and ideal system due to variable delays, describes the same thing.

The total transfer of the model is shown in figure 6.14. Actually the delay ΔT^m_k influences the controller output delay also, therefore we substitute $\Delta T_k = \Delta T^m_k + \Delta T^u_k$. The computation delay is now represented by block $\Lambda(s)$, but we can also add the delay to delay ΔT_k. This would results in the same analysis.

7. SIMULATION

The model that is discussed in chapter 6 can be formulated into a Simulink simulation model. This enables time domain simulation of controller systems that are exposed to variable measurement and actuator delays. We will describe the set-up of the simulation in section 7.1. The results of different simulation examples are discussed in section 7.2. At last we evaluate the simulation model in section 7.3.

7.1. SET-UP

With the set-up of the simulation model we will describe how we simulate the variable measurement delay and later we introduce the variable actuator delay. The model shown in figure 6.14 is considered and converted to a simulation model.

Before we describe the simulation we consider the disturbance $d_e(k)$. This signal adapts the error samples that go into the controller. The approximation method of equation 6.1 and 6.2 assume that the signal $e(t)$ is smooth and can be differentiated. Because the signal $e(t)$ consists of two signals $y(t)$ and $r(t)$, these signals are restricted to this property as well. In the simulation it is common to take a step function as reference signal $r(t)$, which is in contradiction with the definition of the model. If only the output $y(t)$ is sampled, we can choose any form of signal for the discrete time reference input. According to the theory, represented in chapter 6, we have to adapt the sampled signal with a disturbance that is based on the original continuous time derivative of the sampled signal. The analysis of the system stays the same, only the disturbance input will now change the sampled output $y(t)$. Based on equation 6.5 we define this disturbance $d_y(t)$ equation that adjust the sampled output $y(t)$ by

$$d_y(t) = \sum_{k=0}^{k=\infty} \dot{y}(kT)\,\Delta T_k^m\,\delta(t-kT). \tag{7.1}$$

7.1.1. VARIABLE MEASUREMENT DELAYS

We consider the first loop of the model that adjust the ideal sampled values to the real sampled values, based on the measurement delay ΔT^m_k. This loop can be simply converted to a Simulink model, see figure 7.1. Immediately, we see that the disturbance is calculated with the time derivation of $y(t)$. With the random block the variable measurement delays become uniformly distributed. The configuration of the loop can be altered by changing the position of the switches.

The second loop of the model is used to delay the controller output. Figure 7.2 shows the Simulink scheme of this loop. We see that the disturbance is calculated based on the discrete

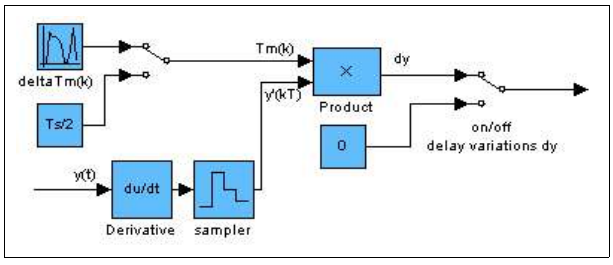

Figure 7.1 Block scheme in of the first loop of the Simulink model.

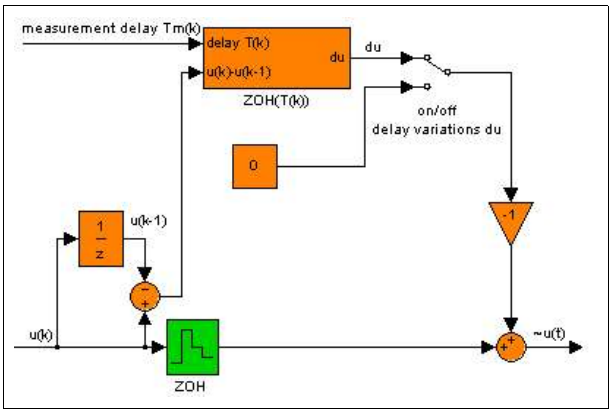

Figure 7.2 Block scheme in of the second loop of the Simulink model.

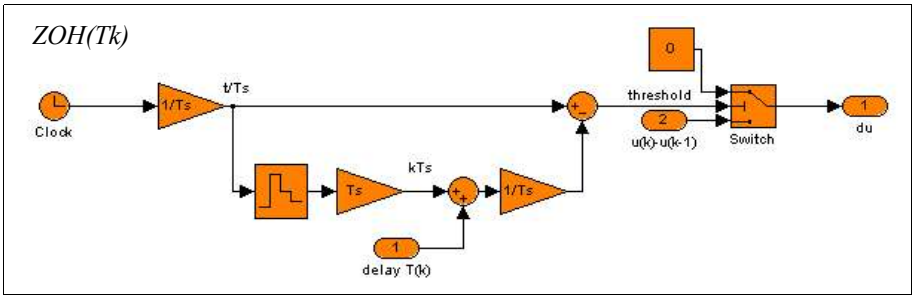

Figure 7.3 Block scheme in Simulink of the controller delay subsystem.

derivative of the controller output. The "controller delay du" subsystem delays the *ZOH* output. This subsystem holds the disturbance output for ΔT^m_k seconds.

In figure 7.3 the subsystem is shown. We use a switch with a threshold which is used to switch between the derivative of the controller output or a zero signal. When the threshold signal is below zero input 2 is passed through, otherwise input 1. Every time instance k the switch has to pass input 2 for a duration of ΔT_k seconds. We calculate a threshold signal based on the time and the delay ΔT_k, therefore we want to create a function that start below zero at every moment kT and passes zero at $k(T+\Delta T_k)$. Figure 7.4 shows the signals of the subsystem. We see that each period the blue line crosses the zero at different moments, due to the random block. By subtracting the green from the red signal we get the blue threshold signal that is used to control the switch. This results in a delayed controller output and is shown in figure 7.5.

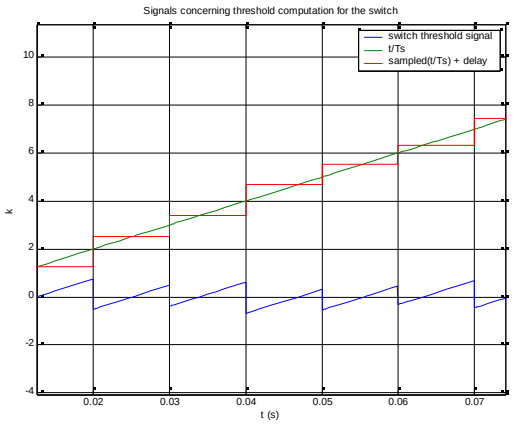

Figure 7.4 Signals concerning threshold computation of the switch.

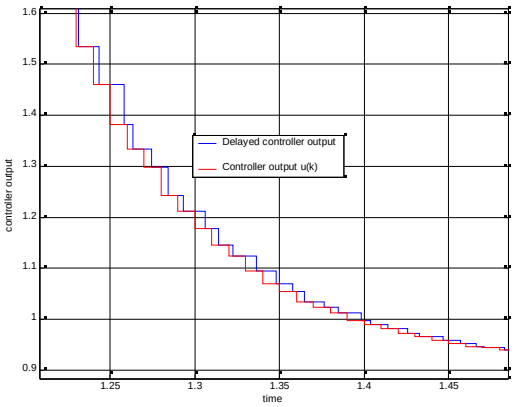

Figure 7.5 The controller output u(k) (red) and the delayed output (blue), due to the disturbance.

7.1.2. Actuator delay and the computation time

Implementation of the actuator and the computation delay is done in the second loop that is shown in figure 7.2. The delay values are added to the measurement delay and is shown in figure 7.6. For the variable actuator delay, we take also a random function. The computational delay is represented by a constant value. Note that the total delay of the *ZOH* output is the sum of all mentioned delays. We consider only semi-synchronous delays and therefore

$$\Delta T^m_k + \Delta T^a_k + \Delta T^c \leq T_s$$

has to hold. For the simulation we will take *Ts/2* as maximum for the measurement as well as for the actuator delay and neglect the computation delay time. Then the definition of semi-synchronous will hold.

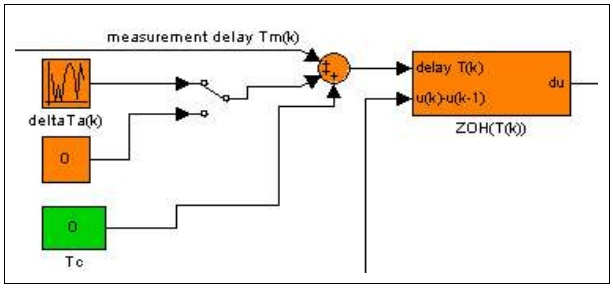

Figure 7.6 Implementing the actuator and the computation delay into the second loop.

7.1.3. THE TOTAL SIMULATION MODEL

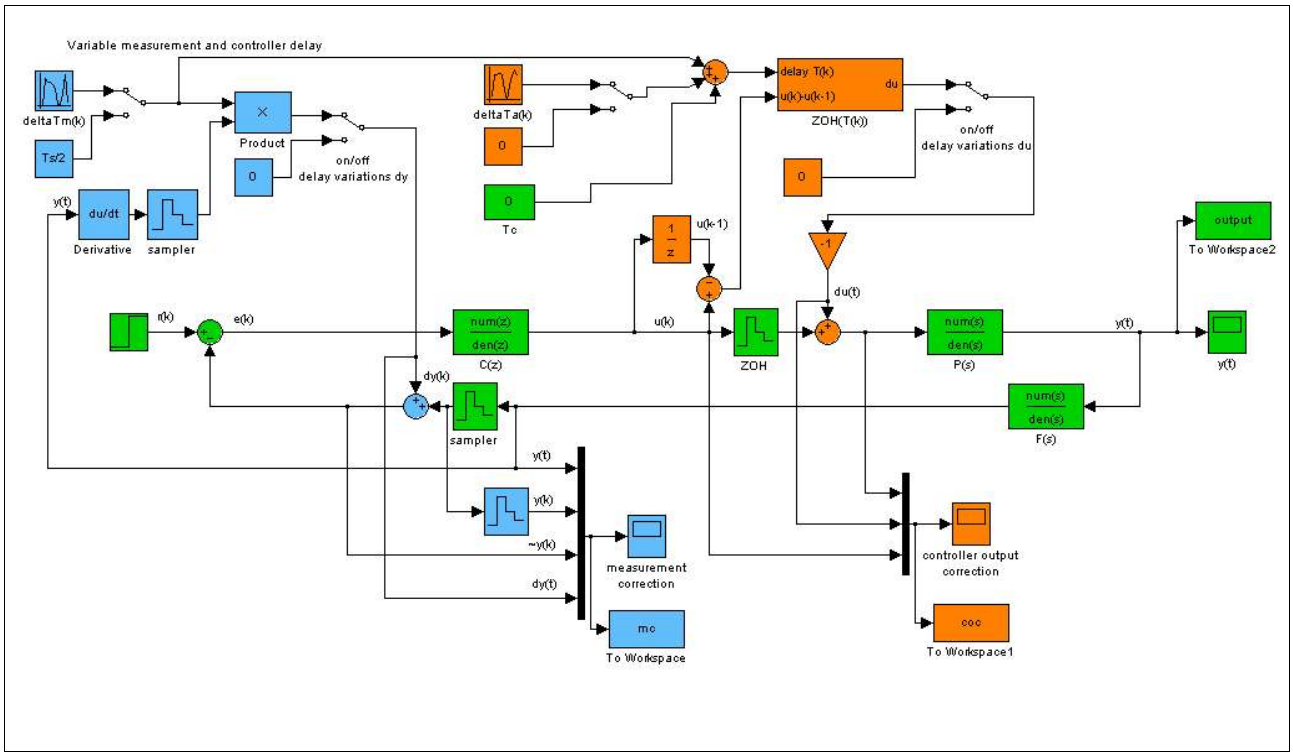

Figure 7.7 Implementation of the model into Simulink.

Figure 7.7 shows the Simulink simulation model. We are able to simulate different systems in the time domain that are exposed to variable delays in the controller loop. The behaviour of the system and the contents of the disturbance signals can be analysed.

7.2. EXAMPLE

We consider the same processes as used in section 5.3. We start with stable plant, so we can see that the simulation disturbs the system as predicted. The first plant and controller are given by

$$P_1(s)=\frac{1}{(s+5)}, \quad \text{and} \quad C_1(z)=\frac{25T}{8}\frac{z+1}{z-1}.$$

We use a sampling period of 0.2 seconds in this case. Figure 7.8 shows the step response of this implementation in an ideal (without variable delays) situation and with variable delays. A variable delay realisation is set with the seed property, so we are able to change the behaviour of the random function. We see that the output does not vary very much and we can say that the system is robust to these delay variations. If we look to the measurement correction, in figure 7.9, we see that the

content of the disturbance signal is very small compared to the output signal. However, this disturbance adjust the samples to the values of the future values and this is clearly shown. Also the controller delay disturbance, in figure 7.10, is very small compared to the controller output. We see that the output of the controller is delayed every period with another delay duration.

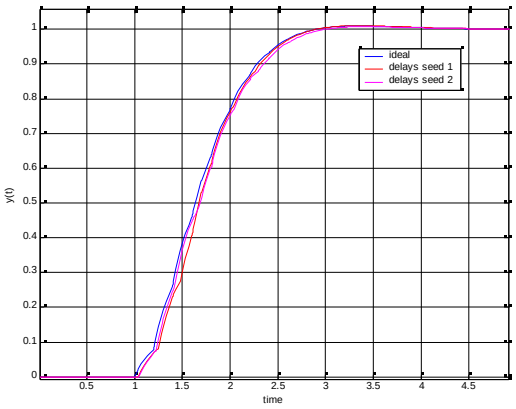

Figure 7.8 Step response of P1 and C1.

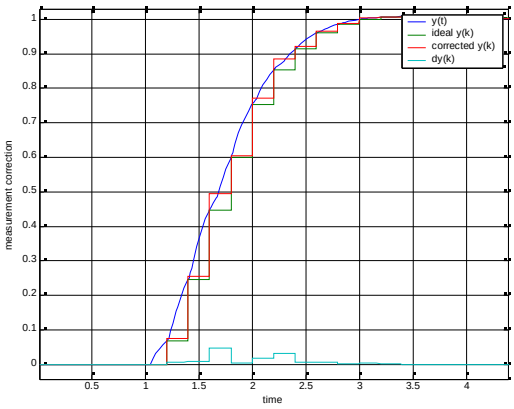

Figure 7.9 Signals concerning measurement correction.

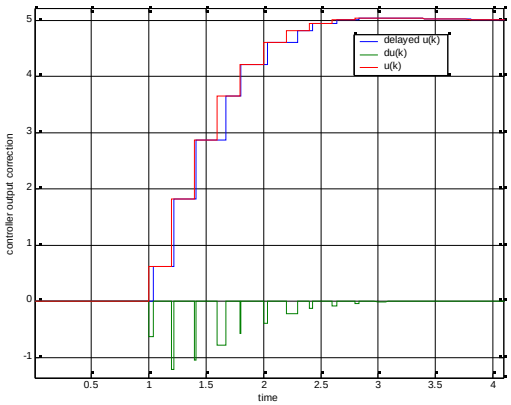

Figure 7.10 Signals concerning controller output delay.

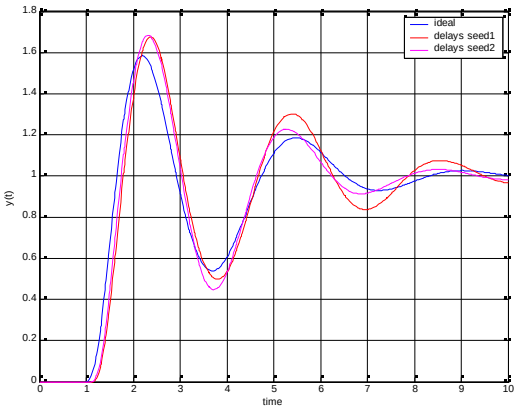

Figure 7.11 Step response of P2 and C2.

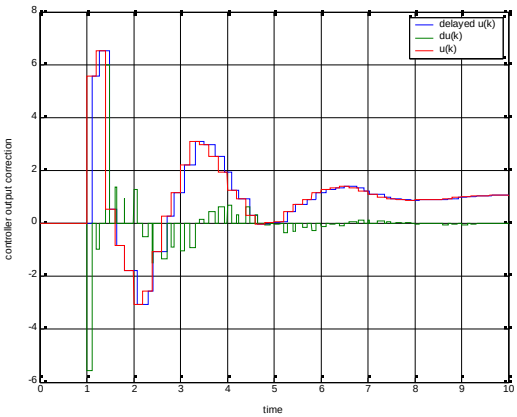

Figure 7.12 Signals concerning controller output delay.

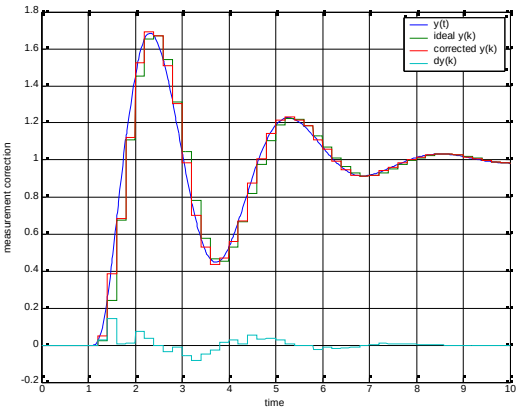

Figure 7.13 Signals concerning measurement correction.

We have shown in chapter 5 that an unstable plant reveals much more difference concerning sampling frequency variations. We take the example of an unstable plant that is used in section 5.3. The second plant and controller are given by

$$P_2(s)=\frac{1}{(s^2+1)}, \quad \text{and} \quad C_2(z)=\frac{5.5625\,z^3-4.3125\,z^2-5.3125\,z+4.5625}{z^3-z^2}.$$

The sampling period of this system is 0.2 seconds. Figure 7.11 shows different step responses of the system based on different variable delay realisations. We see that the responses are significant different with each delay realisation. However, the content of the disturbances, shown in figure 7.12 and 7.13, we see that it is small compared to the system signals.

We consider the same unstable plant, but with another controller. This controller regulates the plant just enough to stabilize it. The sampling period is chosen at 0.4 seconds. The third plant and controller are given by

$$P_3(s)=\frac{1}{(s^2+1)}, \quad \text{and} \quad C_3(z)=\frac{5.5556\,z^3-2.8889\,z^2-4.6667\,z+3.7778}{z^3-0.3333\,z^2-0.5556\,z-0.1111}.$$

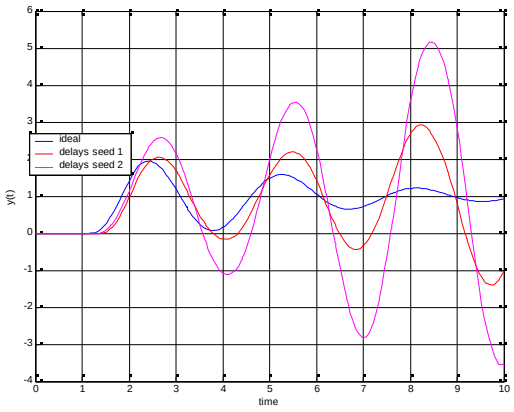

Figure 7.14 Step response of P3 and C3.

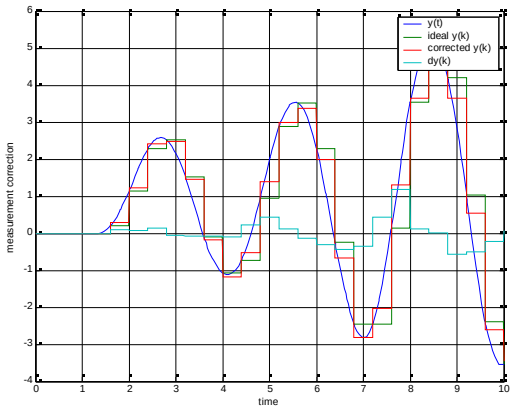

Figure 7.15 Signals concerning measurement correction.

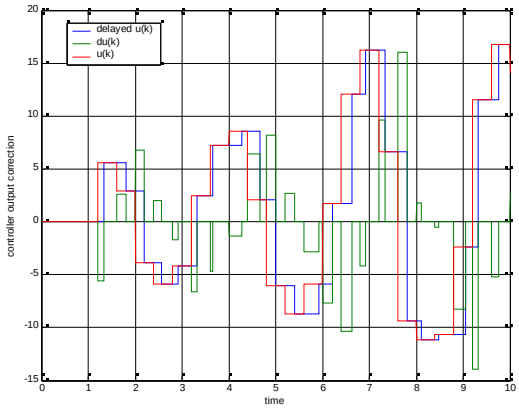

Figure 7.16 Signals concerning controller output delay.

In figure 7.14 the response of this system to a step function is shown for different situations. We see that the controller stabilizes the plant in the ideal situation. However, when the systems is exposed to variable delays it becomes unstable. Figure 7.15 and 7.16 shows that the disturbance content increases when the controller system becomes unstable or as the signals rapidly changes. However, the disturbance of the measurement (see figure 7.15) is still small compared to the

sampled signal. The content of the disturbance of the actuator, shown in figure 7.16, is in this case very large compared to the controller signals. We conclude that the influence of the actuator delay is the most and the controller system cannot stabilize the plant any more.

7.3. CONCLUSION

The model that is described in chapter 6 can be well described in a simulation model. In this chapter we used the simulation application Simulink that comes with Matlab. We are able to analyse the behaviour of the controller system when it is exposed to variable measurement and actuator delays.

However, we see in the simulation results that the output can be different with the use of different variable delay realisations (seeds). Figure 7.17 shows different realisations that can be used to simulate a controller system. This opens the discussion how we can describe the worst-case delay realisation of a particular system. We clearly can not define a variable delay realisation that is the worst-case situation for all controller systems. The worst-case variable delay depends on the dynamics of the system. So each particular system will have its own worst-case variable delay realisation. We could find a worst-case realisation by calculation of the delay value for each sampling instant *k* in which the output of the controller system will have the maximum costs.

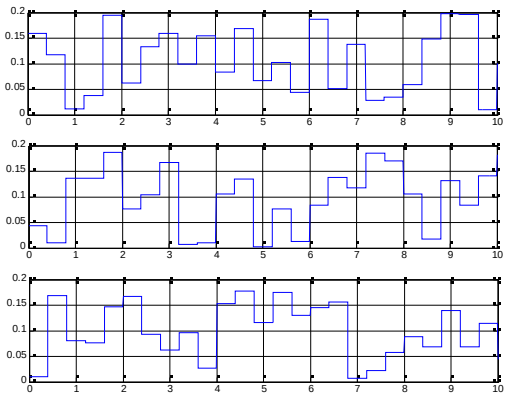

Figure 7.17 Different variable delay realisations.

8. FREQUENCY DOMAIN ANALYSIS

As mentioned in chapter 4, sampled-data systems are mostly analysed in the frequency domain, because both discrete and continuous time parts can be analysed at the same time and a lot of analysis methods are already developed in this domain. Therefore, we make the step to translate the model, described in chapter 6, to the frequency domain. The model introduces two input disturbance signals that are described in the time domain by equation 6.5 and 6.7. Section 8.1 translates these signals to their Laplace equivalents. Describing the response of the model is a bit more complex, due to the introduced loops. In section 8.2 we develop the response of the system with simplified disturbance loops.

8.1. SPECIAL DISTURBANCE INPUTS

The special disturbance signals of the model are given by equation 6.5 and 6.7. We translate these equations to their s-domain equivalent by using the definition of the Laplace integral. Beginning with disturbance $d_e(t)$ we get

$$D_e(s)=\mathcal{L}\{d_e(t)\}=\int_{-\infty}^{\infty} d_e(t)e^{-st}dt=\int_{-\infty}^{\infty}\{\sum_{k=0}^{\infty}\dot{e}(kT)\Delta T_k^m\delta(t-kT)\}e^{-st}dt. \tag{8.1}$$

Rearranging the equation gives

$$D_e(s)=\sum_{k=0}^{\infty}\dot{e}(kT)\Delta T_k^m\int_{-\infty}^{\infty}\delta(t-kT)e^{-st}dt. \tag{8.2}$$

Using the sifting property of the Dirac function gives us

$$D_e(s)=\sum_{k=0}^{\infty}\dot{e}(kT)\Delta T_k^m e^{-skT}. \tag{8.3}$$

Notice, that the equation can be translated to the z-domain by substituting $z=e^{sT}$, so the equation becomes

$$D_e(z)=\dot{e}(kT)\Delta T_k^m z^{-k}. \tag{8.4}$$

To get an idea of the spectrum of disturbance $D_e(s)$, figure 8.1 shows two different spectra based on

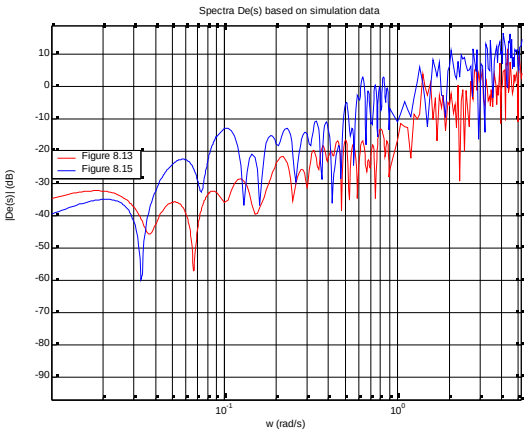

Figure 8.1 Spectrum of De(s) based on simulation data.

the results of the simulations discussed in chapter 7. A stable (red line), of example P_2 and C_2, and an unstable (blue line), of example P_3 and C_3, controller system is shown in this figure. The

frequency spectrum of the two disturbances do not show a significant difference, because the variable delay in equation 8.4 influences the gain of the spectrum and this value will not become bigger than the period time T for all k.

Transformation of the disturbance $d_u(t)$ gives us

$$D_u(s)=\mathscr{L}\{d_u(t)\}=\int_{-\infty}^{\infty} d_u(t)e^{-st}dt=\int_{-\infty}^{\infty}\left\{-\sum_{k=0}^{\infty}\Delta u(k)\big(\varepsilon(t-kT)-\varepsilon(t-kT-\Delta T_k)\big)\right\}e^{-st}dt. \tag{8.5}$$

Rearranging the equation and using the property of the step function $\varepsilon(t)$ to adapt the integral borders, we get

$$D_u(s)=-\sum_{k=0}^{\infty}\Delta u(k)\int_{kT}^{kT+\Delta T_k} e^{-st}dt=-\sum_{k=0}^{\infty}\Delta u(k)\int_{0}^{\Delta T_k} e^{-st}dt. \tag{8.6}$$

Solving the integral results in

$$D_u(s)=\sum_{k=0}^{\infty}\Delta u(k)\left(\frac{1-e^{-s\Delta T_k}}{s}\right) \tag{8.7}$$

We see that the last expression in the equation is very similar to the equation of a zero-order-hold function, but each sampling instant it gets another impulse response due to the variable delay $\Delta T^m{}_k$. Every sample this function has to hold the control output signal $\Delta u(k)$ for a specified delay time. Based on simulation data, described in chapter 8, we show two different spectra, based on the same examples of figure 8.1, of disturbance $D_u(s)$ in figure 8.2. As expected, the spectra looks a bit like the zero-order-hold frequency spectrum. However, in contrary to figure 8.1 we see a significant difference between the stable and unstable controller system. This is caused by controller output signal $\Delta u(k)$. Equation 8.7 shows that this signal determines the gain of the spectrum and that the signal will grow when the system becomes unstable. This implies that the gain of the spectrum will grow too.

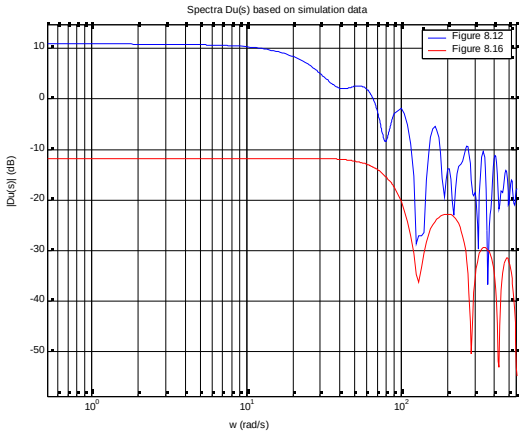

Figure 8.2 Spectrum of Du(s) based on simulation data.

8.2. RESPONSE OF THE MODEL

This section describes the model, described in chapter 6, in the frequency domain. Because of the special disturbance inputs, complex loops are introduced in the system. However, to retrieve the correct transfer function of the model, we have to take these loops into account. Chapter 5.1 describes the response in the frequency domain of a standard sampled-data system. We use the same techniques to derive the frequency response [14]. In section 8.2.1 we describe the response of a

simplified model in which the complex loops are replaced by a simple transfer function. Section 8.2.2 discusses the complexity of the transfer functions of both loops.

8.2.1. SIMPLIFIED MODEL

We have simplified the model in order to derive the frequency response of the developed model of chapter 6. The complex loops, of the disturbances, are replaced by a simple transfer functions. Also, we consider that the output *y(t)* is sampled instead of the error *e(t)*. This implies that the reference will be discrete as well. The considered model is shown in figure 8.3. In fact, these kind of models are very common in practice. Normally the reference is created in the computer itself and the output of the process is sampled.

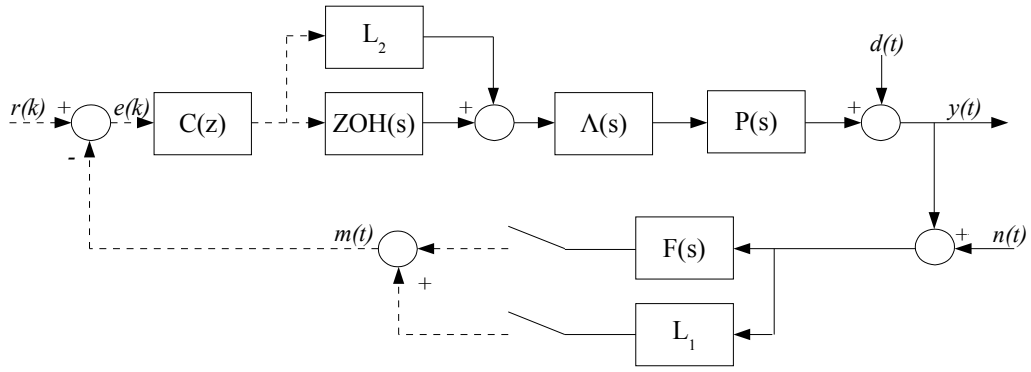

Figure 8.3 Sampled-data system model with L_1 and L_2 as the complex loops of the disturbances.

In appendix C the derivation of this sampled-data system model is given in more detail. This sampled-data system does not have a transfer function, therefore we the discrete fundamental output response of the sampled-data is given by

$$Y_f(s)=\frac{C(e^{st})\frac{1}{T}(ZOH(s)+L_2(s))\Lambda(s)P(s)}{1+C(e^{st})\frac{1}{T^2}\big(F(s)+L_1(s)\big)\big(ZOH(s)+L_2(s)\big)\Lambda(s)P(s)}\big(R(e^{st})-N(s)\big) + \frac{1}{1+C(e^{st})\frac{1}{T^2}\big(F(s)+L_1(s)\big)\big(ZOH(s)+L_2(s)\big)\Lambda(s)P(s)}D(s). \tag{8.8}$$

This fundamental response is very similar as the derived response described in section 4.1. We are able to derive the sensitivity transfer functions for this system as well. So, the fundamental sensitivity function becomes

$$S_f(s)=\frac{1}{1+C(e^{st})\frac{1}{T^2}\big(F(s)+L_1(s)\big)\big(ZOH(s)+L_2(s)\big)\Lambda(s)P(s)}. \tag{8.9}$$

and the complementary sensitivity becomes

$$T_f(s)=\frac{C(e^{st})\frac{1}{T}(ZOH(s)+L_2(s))\Lambda(s)P(s)}{1+C(e^{st})\frac{1}{T^2}\big(F(s)+L_1(s)\big)\big(ZOH(s)+L_2(s)\big)\Lambda(s)P(s)}. \tag{8.10}$$

8.2.2. TRANSFER OF L_1 AND L_2

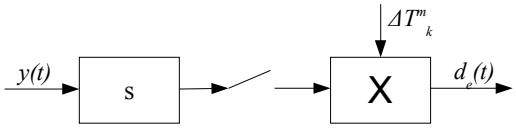

Figure 8.4 Transfer L_1.

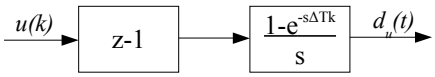

Figure 8.5 Transfer L_2.

Both loops are complex due to the fact that it contains a variable parameter that changes each sampling period. Figures 6.12 and 6.13 show both loops of the model. Note, that we sample the output *y(t)* and in chapter 6 the error *e(t)* is considered as sampled signal. We can convert these loops to the frequency domain based on equations 8.3 and 8.7 (see figure 8.4 and 8.5). Because of the variable delays the loop becomes time-variant, therefore we cannot describe the loops by time-invariant transfer functions. In order to do further calculation we have to decided how to treat the variable delay in these equations. A possible approach could be to treat the variable delay as a constant, a continuous time signal or a stochastic variable. Analysis of a constant variable is not the interest of this study. However, it would be a simple multiplication in the s-domain. We see that the time domain multiplication will be translated to a convolution in the frequency domain. This complicates the equations. A stochastic approach seems to suitable, because a lot of developed methods can be used. However, further study of this subject is considered as future work and will not be treated in this report.

9. CONCLUSIONS

Embedded control systems are complex, because they consist of different subsystems that are linked together; mechanics, electronics and software. More often, controllers are implemented on a processor. We see that the simplest embedded control system contains a multitasking realtime operating system. Software engineers cannot guarantee a deterministic behaviour of the software platform and that results in different variable delays that are introduced in the controller loop. Performance and stability becomes an issue and the control engineer cannot deal with this kind of variable delays in the design. Mostly, the present controller algorithms have strong timing requirements. This results in implementation problems of controller algorithms that rises with the development of embedded controller systems. We have to understand the operation of these complex systems more better. A lot of problem statements were given in the introduction and we will discuss them separately.

With the design of embedded systems there needs to be more cooperation between the software and control. Only then an optimal solution can be found. To create a more robust controller we can think of a feedback loop from the controller to the scheduler and from the scheduler to the controller. This gives the opportunity to adjust the scheduling of the executed tasks and the parameters of the controller algorithm to the dynamical situation of the embedded control system. When timing is crucial the scheduler can take a higher sampling frequency for the controller task. However, when the controller is not sensitive to variations, for example when it is in steady state, than the scheduler can take a lower sampling frequency and more processing resources are available for other tasks. The controller have to compensate its calculations based on the information of the scheduler, for example when it switches to another sampling frequency.

When we consider a controller algorithm we can distinguish three delay types: measurement ,computation and actuator update delays. Computational delay is the execution of the algorithm and is assumed to be constant. Measurement and actuator update delay vary randomly each sampling period. We only consider semi-synchronous delays, therefore all the delays together cannot become bigger than the sampling period of the controller system.

A first analysis of the system is to determine the global sensitivity to sampling period variations. With the SSF (sensitivity sampling frequency) function we calculate how much a given transfer function changes, when the controller system is exposed to sampling period variations. The transfer function is obtained by the sampled-data method, so that both discrete and continuous parts of the system are taken into account simultaneously. Globally, we see that the sensitivity increases when the system is excited with a signal that contains higher frequencies and when the sampling frequency of the controller decreases.

The introduced variable delays in the controller system, makes the system time-varying. All theory for analysis of time-invariant systems cannot be used directly. A method to analyse the controller system in the time-invariant domain is to model the behaviour of the variable delays as external disturbance inputs to the system. These disturbances actually represent the error between the ideal and the practice situation of the controller system. Analysis of the disturbances tells that the contents of the signal depends on the derivative of the signal that is disturbed. So, the influence of the disturbance on the system grows proportional with the derivation of the signals and can make the performance of the controller system worse. This developed model can be easily converted to a time domain simulation tool; such as Simulink. Analysis of a system described in the z- and s-domain can be performed under various variable delay conditions. This gives more insight into the behaviour of the controller system that is exposed to variable delays.

The introduced disturbance inputs depends on the derivative of the signals in the controller system. This implies that there exists loops from the system signals to the disturbance inputs. Due to the time-varying behaviour of the disturbances, these loops become very complex. These loops

influence the performance of the controller system. In the worst-case situation the system can become unstable. Analysing the effect of variable delays in a controller system can be done by analysing the developed disturbance inputs. They represent the error between the ideal and the practice situation of the controller system. To describe the disturbance in the frequency domain is very complex and needs more research. However, we are able to determine the frequency spectra based on simulation data. We see that when the derivative of the signals in the system becomes bigger, the contents of the disturbance content of the actuator delay grows. Concluding that the actuator delay has the most effect on the system in this situation. The content of the disturbance of the measurement delay stays relatively small. Analysis of these complex loops helps to get more insight into the controller system and eventually that can lead to an improved controller design.

Mostly, the present controller algorithms that are implemented in embedded systems cannot be simply adapted in order to achieve an improved embedded controller system. For this, embedded systems are too complex. We have to consider both the software and controller design simultaneously. Only then we are able to develop methods to design “optimal” embedded control systems. Different approaches are possible in order to enridge the embedded control system, for example adapting the sampling frequency online, feedback scheduling, stability analysis of the complex loops of the disturbances and event driven control. When the sampling frequency is adjusted, this can be based on the disturbance contents that represent the error between the ideal and the real situation of the controller. Actually, this is a feedback loop from the controller signals to the sampling unit. The dynamic behaviour of the system when varying delays are introduced, can be controlled with this feedback loop. Also, the software scheduling algorithm is able to do optimal scheduling, so that the controller tasks that run on the system meet their required performance. Off course, the strong timing requirements can be stretched when the controller algorithm can adapt the algorithm to these variable delays. This interaction between the software scheduler and the controller algorithm has to improve the controllability of the total embedded system.

However, there is a lot of work that still has to be done in the world of embedded systems. We need to understand these systems to be able to design the optimal embedded controller system. Every method or model will open new opportunities, insights and possible solutions to the problem. For the controller and software science this becomes a very nice embedded playing ground of research.

10. RECOMMENDATIONS

During the study different ideas came up for interesting design approaches or new study directions. These are described in this chapter. In section 10.1 we discuss an adjustment of the measurement value to improve the performance of the embedded controller system. This can be done in the continuous as well as in the discrete domain. Section 10.2 discusses the possibility to adjust the sampling period and section 10.3 discusses the feedback scheduler to improve embedded control systems.

10.1. ADJUSTING MEASUREMENT VALUES

We consider a simple control system scheme shown in figure 10.1. With the introduction of a zero-order-hold a delay of *T/2* is introduced at the input of the plant *P(s)* [14]. This phenomenon is shown in figure 10.2. Each value of the controller output *u(kt)* is held constant, by the *ZOH*, until the next controller value comes available. This means that the continuous time value of *u(t)* consists of steps that, on the average, lag *u(kT)* by *T/2* (dashed line). In fact, the plant is excited with a delayed controller input signal. However, this delay is not compensated in the controller system. Based on the theory of the model described in chapter 6, we are able to compensate this delay by using a disturbance input that is placed at the sampling unit. So, the measurement samples are adjusted in order to compensate the introduced delay of *T/2*.

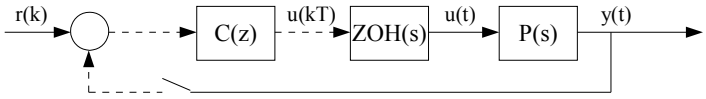

Figure 10.1 Simple control system scheme.

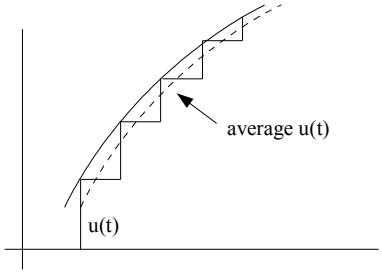

Figure 10.2 Due to the ZOH a delay of T/2 is introduced (dashed line).

This adjustment is simulated with different plants and controllers. The results are shown in figure 10.3 – 10.5. In each case the system responses improves (blue line). However, to adjust the measurement values equation 6.2 is used (for *y(t)*) and depends on the continuous time derivative of the output *y(t)*. This complicates the implementation of this method. The adjustment can be approximated in the discrete domain by a discrete derivative of the sampled output *y(k)* and can be

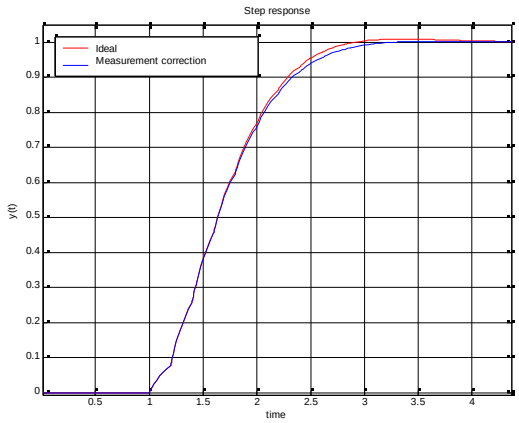

Figure 10.3 Stable process response (T=0.2).

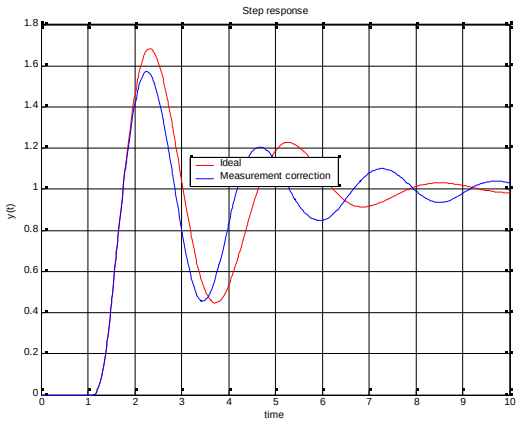

Figure 10.4 Unstable process step response (T=0.07).

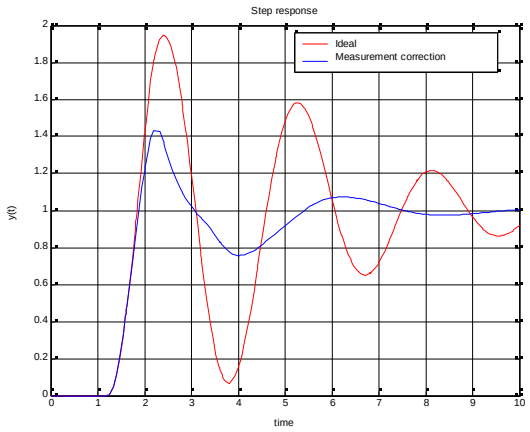

Figure 10.5 Unstable process step response (T=0.7).

easily be implemented in an existing controller algorithm. The equation that adjust the sampled value at the beginning of the controller algorithm becomes

$$y^*(k)(k) = y(k) + \frac{(y[k] - y[k-1])T}{2}. \tag{10.1}$$

Figure 10.6 and 10.7 shows different responses and we see again that the response of the system is improved. It can be a method to improve some embedded controller systems. Figure 10.8 shows the set-up which is used to implement the discrete compensation in the simulation.

This method needs research and is considered as future work. We see already that the simulation results are very promising. To compensate the delay we can also adjust the *ZOH* output. This is studied before and is called half-order-hold [21].

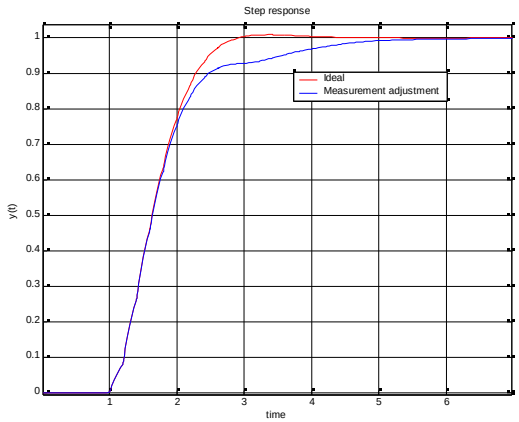

Figure 10.6 Stable process step response (T=0.2).

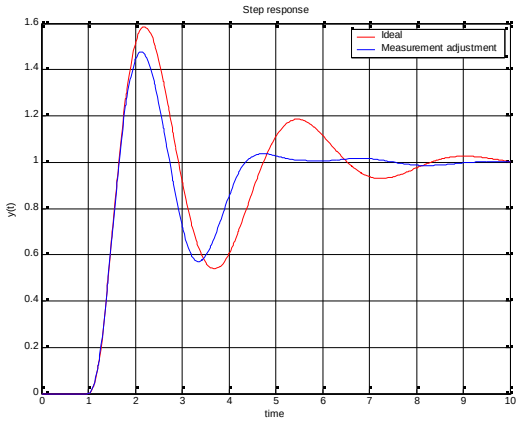

Figure 10.7 Unstable process step response (T=0.2).

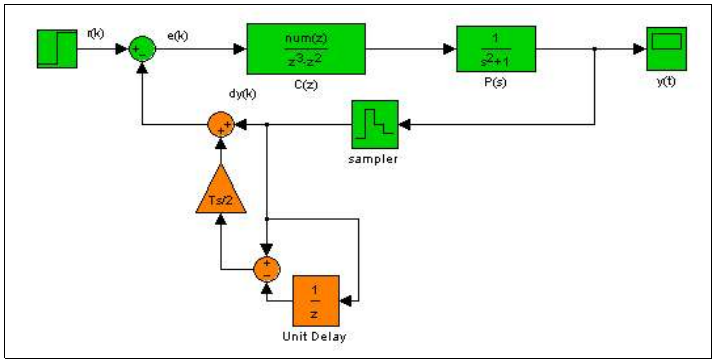

Figure 10.8 Discrete measurement compensation.

10.2. ADJUSTING SAMPLING FREQUENCY BASED ON DISTURBANCES CONTENT

The disturbance signals represent the error between the intended and the practice situation. We want to feedback the contents of these signals in order to adapt the sampling frequency. In fact, the represented error has to be as small (or within some range) as possible by controlling the sampling frequency. When the error becomes bigger the sampling frequency will be increased and when the error becomes less the sampling frequency is decreased. This is the basic idea that is developed. Figure 10.8 shows an example of a controller scheme with feedback and a controller that regulates the sampling frequency. In this case the sampling frequency will be low when the system is in steady state or the delays become very small (see equations 6.5 and 6.7).

The "controller T" calculates based on the disturbance signals the optimal sampling frequency of the control system. In fact, when the output signal changes rapidly we need more samples than when it changes slowly. The disturbance signals represent the status of these signals. Note that the controller *C(z)* is a event-driven controller and computes only the next actuator output when a sample is taken.

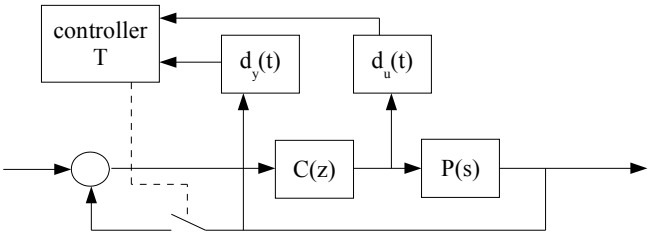

Figure 10.8 Controller scheme with sampling frequency regulation based on disturbance signals.

10.3. FEEDBACK SCHEDULING

The overall performance of an embedded control system depends on the cooperation between the software and the control engineers. Because control systems are more and more implemented on multitasking computing platforms, scheduling becomes very important [22]. A feedback scheduler that retrieves information about the status of a controller algorithm task can be important to control the introduced delays of the software. For example, a feedback scheduler can calculate a cost function which represents the optimal sampling frequency of the controller task [10]. The scheduler can also consider the disturbance contents to determine the sampling frequency of a particular task. However, it becomes very important to have a feedback loop to and from the software scheduler and the controller task in order to design an optimal embedded controller that is implemented on a multitasking software platform.

APPENDIX A

DERIVATION FREQUENCY RESPONSE SAMPLED-DATA SYSTEMS

For the derivation of the response in the frequency domain of sampled-data system we use the methods that are presented in the book Digital control of dynamical systems [14]. A sampled-data system consists of discrete and continuous time parts. To connect these different parts a sampler and a hold circuit are used. These elements make the total system nonlinear and time-variant. Analysing these kind of systems is mostly done in the frequency domain. The discrete as well as the continuous time part can be easily represented in the frequency domain. However, the response of a sampled-data system has to be derived. We will see that not every system has a transfer function.

We begin with the description of a sampled signal. A signal *r(t)* is sampled. We use the impulse modulation model to describe the sampled version *r*(t)* of the signal *r(t)*. The description of the sampled signal becomes

$$r^*(t)=r(t)\sum_{k=-\infty}^{\infty}\delta(t-kT). \tag{A.1}$$

We see that the sampled signal *r*(t)* is a product of *r(t)* and a train of impulses. For further insight into the process of sampling we use the Fourier analysis and describe the pulse train by its Fourier series. The pulse train can be represented by

$$\sum_{k=-\infty}^{\infty}\delta(t-kT)=\sum_{n=-\infty}^{\infty}C_n e^{j(\frac{2\pi n}{T})t}, \tag{A.2}$$

where the Fourier coefficients, C_n, are given by the integral over one period as

$$C_n=\frac{1}{T}\int_{-T/2}^{T/2}\sum_{k=-\infty}^{\infty}\delta(t-kT)e^{-jn(\frac{2\pi t}{T})}dt. \tag{A.3}$$

Only the term in the sum of impulses that is in the range of the integral is at the origin of *δ(t)*, so we can reduce the integral to

$$C_n=\frac{1}{T}\int_{-T/2}^{T/2}\delta(t)e^{-jn(\frac{2\pi t}{T})}dt. \tag{A.4}$$

With the sifting property of *δ(t)*, the function has only an output when *t=0*, it becomes easy to integrate and results into

$$C_n=\frac{1}{T}. \tag{A.5}$$

Thus we have derived the representation of the sum of impulses as a Fourier series:

$$\sum_{k=-\infty}^{\infty}\delta(t-kT)=\frac{1}{T}\sum_{n=-\infty}^{\infty}e^{j(\frac{2\pi n}{T})t}. \tag{A.6}$$

We define $w_s=2\pi/T$ as the radian sampling frequency and substitute A.5 into A.1 using w_s, and we take the Laplace transform of the output of the mathematical sampler:

$$\mathscr{L}\{r^*(t)\}=\int_{-\infty}^{\infty} r(t)\left\{\frac{1}{T}\sum_{n=-\infty}^{\infty} e^{jnw_s t}\right\}e^{-st}\,dt. \tag{A.7}$$

We integrate the sum and combine the exponentials:

$$R^*(s)=\frac{1}{T}\sum_{n=-\infty}^{\infty}\int_{-\infty}^{\infty} r(t)e^{-(s-jnw_s)t}\,dt. \tag{A.8}$$

The integral here is the Laplace transform of *r(t)* with only a change of variable of the frequency and the result can therefore be written as

$$R^*(s)=\frac{1}{T}\sum_{n=-\infty}^{\infty} R(s-jnw_s), \tag{A.9}$$

where *R(s)* is the Laplace transform of *r(t)*. We see that the frequency spectrum of a sampled signal consist of a never ending train of sidebands. This is also the place where aliasing comes in. There is overlap between the spectral copies and that is called aliasing. Therefore it is important to filter the continuous time signal before it is sampled. The frequency spectrum copies will not have any overlap and aliasing is avoided, when the filter is of an appropriate high order.

Having a model of the sampling operation, we will now consider the hold operation to complete the sample-and-hold. The hold operation will take the impulses that are produced by the mathematical sampler and produce a piecewise constant output over the sampling intervals. We can mathematically describe the hold function *h(t)*, using the step function *ε(t)*, by

$$h(t)=\varepsilon(t)-\varepsilon(t-T). \tag{A.10}$$

The required transfer function is the Laplace transfer of *h(t)* and becomes

$$ZOH(s)=\mathscr{L}\{h(t)\}=\int_0^{\infty}\{\varepsilon(t)-\varepsilon(t-T)\}e^{-st}\,dt=\frac{\left(1-e^{-sT}\right)}{s}. \tag{A.11}$$

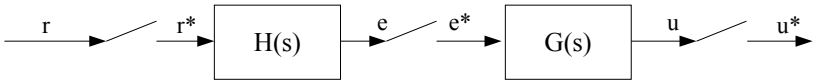

Figure A.1 A cascade of samplers and filters.

All sampling elements have been discussed. We begin with the block diagram analysis of sampled-data systems. Figure A.1 shows a block scheme of samplers and filters and we represent this scheme by a sampled-data transfer function in the Laplace domain. The first equations of the system leads to

$$\begin{aligned} E(s)&=R^*(s)H(s),\\ U(s)&=E^*(s)G(s). \end{aligned} \tag{A.12}$$

The result of impulse modulation of continuous time signals like *e(t)* and *u(t)* is to produce a series of sidebands as given in A.9. If the transform of the signal to be sampled is a product of a transform that is already periodic of period $2\pi/T$, and one that is not, we can show that the periodic transform comes out as a factor of the result. This is the most important relation for the block diagram analysis

of sampled-data systems. This means that if we sample the equation of A.12 they become

$$E^*(s)=(R^*(s)H(s))^*=R^*(s)H^*(s),$$
$$U^*(s)=(E^*(s)G(s))^*=E^*(s)G^*(s). \tag{A.13}$$

We can prove this by using A.9, if *U*(s)=E*(s)G(s)* we have

$$U^*(s)=\frac{1}{T}\sum_{n=-\infty}^{\infty}E^*(s-jnw_s)G(s-jnw_s).$$

However, the definition of *E*(s)* is

$$E^*(s)=\frac{1}{T}\sum_{n=-\infty}^{\infty}E(s-jnw_s),$$

so that

$$E^*(s-jnw_s)=\frac{1}{T}\sum_{n=-\infty}^{\infty}E(s-jkw_s-jnw_s).$$

In this equation we can substitute $k = l - n$ to get

$$E^*(s-jnw_s)=\frac{1}{T}\sum_{l=-\infty}^{\infty}E(s-jlw_s)=E^*(s).$$

In other words, because *E** is already periodic the equations of A.13 are correct. However, if the equation $U(s) = E(s)G(s)$ was the case then $U^*(s) \neq E^*(s)G^*(s)$, but $U^*(s) = (E(s)G(s))^*$. The periodic character becomes crucial. We are now able to give the final equation of the total transfer by using A.12 and A.13:

$$U^*(s)=R^*(s)H^*(s)G^*(s) \tag{A.14}$$

The final result that we require is that we can find the corresponding z-transform of a sampled-data transform, such as *U*(s)*, by

$$U(z)=U^*(s)|_{z=e^{sT}}. \tag{A.15}$$

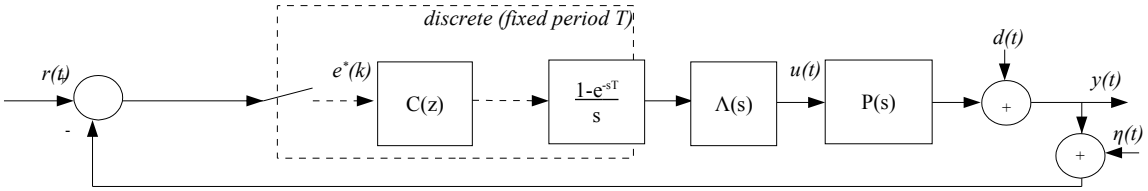

Figure A.2 Scheme of a sampled-data system.

These rules of analysis are illustrated with the derivation of the sampled-data block scheme shown in figure A.2. We can write the relations in the Laplace domain as

$$E(s)=R(s)-Y(s)-N(s), \tag{A.16}$$

$$Y(s)=E^*(s)C^*(s)\left(\frac{1-e^{-sT}}{s}\right)\Lambda(s)P(s)+D(s). \tag{A.17}$$

The usual idea is to relate the discrete output *Y** to the discrete input *R**. Suppose that we sample each of these equations. The equations become

$$E^*(s)=R^*(s)-Y^*(s)-N^*(s), \tag{A.18}$$

$$Y^*(s)=\left(E^*(s)C^*(s)\left(\frac{1-e^{-sT}}{s}\right)\Lambda(s)P(s)+D(s)\right)^*. \tag{A.19}$$

Equation A.19 can be simplified by extracting all periodic transforms:

$$Y^*(s)=E^*(s)C^*(s)\left(1-e^{-sT}\right)\left(\frac{\Lambda(s)P(s)}{s}\right)^*+D^*(s). \tag{A.20}$$

We see in equation A.20 that we need E^* and not E. Substitution of A.18 into A.20 gives us

$$Y^*(s)=\left(R^*-Y^*-N^*\right)C^*(s)\left(1-e^{-sT}\right)\left(\frac{\Lambda(s)P(s)}{s}\right)^*+D^*(s). \tag{A.21}$$

This sampled-data system has a transfer function, because we are able to extract Y and R in order to create a simple transfer function. We are able to define the total transfer function of the sampled-data system by

$$\begin{aligned}Y^*(s)=&\frac{(1-e^{-sT})C^*(s)\left(\frac{\Lambda(s)P(s)}{s}\right)^*}{1+(1-e^{-sT})C^*(s)\left(\frac{\Lambda(s)P(s)}{s}\right)^*}(R^*(s)-N^*(s))\\&+\frac{1}{1+(1-e^{-sT})C^*(s)\left(\frac{\Lambda(s)P(s)}{s}\right)^*}D^*(s).\end{aligned} \tag{A.22}$$

All different transfer function can be derived from equation A.22. For example, if we want to have the transfer function of the output Y^* to the disturbance input N^*, we make all other inputs equal to zero. Then equation A.22 becomes

$$Y^*(s)=-\frac{(1-e^{-sT})C^*(s)\left(\frac{\Lambda(s)P(s)}{s}\right)^*}{1+(1-e^{-sT})C^*(s)\left(\frac{\Lambda(s)P(s)}{s}\right)^*}N^*(s). \tag{A.23}$$

We define two transfer function: sensitivity and the complementary sensitivity transfer function. These functions are equivalent with the continuous time domain theory. The sensitivity S^* and the complementary T^* function are defined by

$$S^*(s)=\frac{1}{1+(1-e^{-sT})C^*(s)\left(\frac{\Lambda(s)P(s)}{s}\right)^*}, \tag{A.24}$$

$$T^*(s)=\frac{(1-e^{-sT})C^*(s)\left(\frac{\Lambda(s)P(s)}{s}\right)^*}{1+(1-e^{-sT})C^*(s)\left(\frac{\Lambda(s)P(s)}{s}\right)^*}. \tag{A.25}$$

If we use the definition of the sampled signals we get

$$S^*(s)=\frac{Y^*(s)}{D^*(s)}=\frac{1}{1+(1-e^{-sT})C(e^{sT})\frac{1}{T}\sum_{n=-\infty}^{\infty}\left(\frac{\Lambda(s-jnw_s)P(s-jnw_s)}{s}\right)}, \tag{A.26}$$

and

$$T^*(s)=\frac{Y^*(s)}{R^*(s)}=\frac{(1-e^{-sT})C(e^{sT})\frac{1}{T}\sum_{n=-\infty}^{\infty}\left(\frac{\Lambda(s-jnw_s)P(s-jnw_s)}{s}\right)}{1+\left\{(1-e^{-sT})C(e^{sT})\frac{1}{T}\sum_{n=-\infty}^{\infty}\left(\frac{\Lambda(s-jnw_s)P(s-jnw_s)}{s}\right)\right\}}. \tag{A.27}$$

That are enormous complex equations, cause of the infinity sidebands of the sampled signals. However, is we take only the fundamental sideband, $n = 0$, then we can define these two transfer functions as fundamental sensitivity function. So the fundamental equations become

$$S_f(s)=\frac{1}{1+\frac{1}{T}\frac{(1-e^{-sT})}{s}C(e^{sT})\Lambda(s)P(s)}, \tag{A.28}$$

and

$$T_f(s)=\frac{\frac{1}{T}\frac{(1-e^{-sT})}{s}C(e^{sT})\Lambda(s)P(s)}{1+\left\{\frac{1}{T}\frac{(1-e^{-sT})}{s}C(e^{sT})\Lambda(s)P(s)\right\}}. \tag{A.29}$$

These equation can be used for fundamental analysis of a sampled-data system. Besides these functions we can use more harmonics by taking more arguments of the base equations A.26 and A.27 [4].

APPENDIX B

SSF CALCULATIONS

Calculation of the SSF (sensitivity sampling frequency) function needs transfer functions that depend on the sampling period T also. Unfortunately, Matlab only considers fixed sampling periods and cannot deal with symbolic transfer functions. In order to work with symbolic transfer functions, we have written some extra Matlab functions. Table B.1 gives an overview of all functions that are concerned with computing the SSF function. Also the source code of the functions is given after the computation of the SSF function.

Table B.1 Overview of the functions concerned with SSF computation.

Function	Description
tf2s (Hs)	Translate the continuous time transfer function of the Matlab type "sys" to a symbolic transfer function based on the Laplace variable *s*.
tf2zT (Hs, converion)	Translate the continuous time transfer function of the Matlab type "sys" to a symbolic discrete time transfer function based on the z-domain and the sampling period, via an approximation method given by *conversion*.
bodesyms (Ps, Ts)	Plot the bode diagram of the symbolic transfer function, that is only based on the Laplace variable *s*. When the function contains also discrete components, such as exp(st), then Ts can be given in order that the function stops plotting after (1/Ts) *pi.
ssf (Ps)	Computes the SSF function of the symbolic transfer function Ps, which contains the variable *s* and *T*.
ssf_plot (Ps, Cz, Ts, transfer, n)	Plot the bode diagram of the controller system that consists of symbolic transfer plant Ps and symbolic transfer controller Cz. Based on the transfer type, for example the complementary sensitivity, the normalized SSF is computed based on the sampling period Ts and n.

We describe a different example as given in chapter 5.3 and begin with the design of the controller. Our plant is described by

$$P(s)=\frac{1}{s^2-1}.$$

This is an unstable second order plant with a mirrored pole in the imaginary axe. We design a feedback controller that stabilise and regulates the error to zero in the continuous time domain. This situation is mostly done in practice. The controller for this plant becomes

$$C(s)=400\frac{s^2+2s+s}{s(s+10)^2}.$$

We transform the controller to the z-domain with the Tustin approximation (use function *tf2zT*) and the discrete controller becomes

$$C(z)=100T\frac{(z+1)(2z^2-4z+2+2Tz^2-T+T^2z^2+2T^2z+T^2)}{(z-1)(5Tz+5T+z-1)^2}.$$

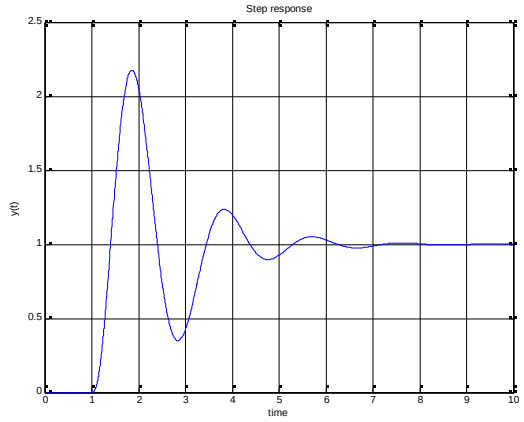

Figure B.1 Step response.

The sampling period in the ideal situation is twenty times the bandwidth of the system, thus the sampling period becomes 0.04 seconds. Figure B.1 shows the step response of the system and we see that the controller stabilises the plant. We compute the SSF of the sensitivity function of this controller system. The sensitivity function that we use is the fundamental discrete sampled-data sensitivity function and is defined as followed

$$S(s)=\frac{1}{1+\frac{1}{T}\frac{(1-e^{-sT})}{s}C(e^{sT})\Lambda(s)P(s)}.$$

This function represents both discrete and continuous time parts of the sampled-data system. The $SSF_s(s)$ is the derivation of the sensitivity function and defined as

$$SSF_S(s)=\frac{\frac{1}{T}\Lambda(s)P(s)}{1+\left(\frac{1}{T}\Lambda(s)P(s)ZOH(e^{st})C(e^{st})\right)^2}\frac{d\left(ZOH(e^{st})C(e^{st})\right)}{dT}$$

We shall plot the normalized SSF function of 1% with function *ssf_plot*. Figure B.2 represents the plot that is calculated by the function. Beware that the plant must be a symbolic continuous time transfer function. It can be easily converted with function *tf2s*. We see that when the sampling frequency changes with 1% the sensitivity function of the controller system will change with 0.02 at maximum. We can conclude that the system is not sensitive to sampling frequency variations.

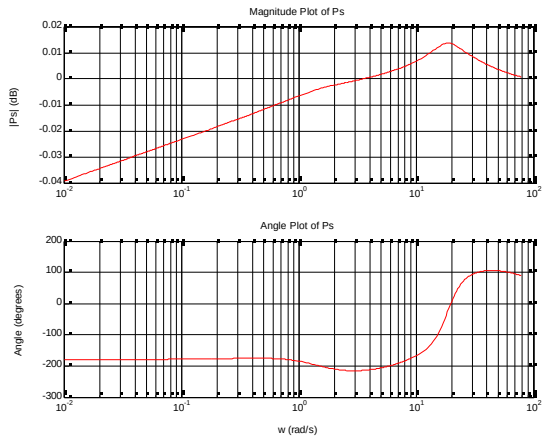

Figure B.2 SSF for sampling period 0.04.

If we are not able to run this algorithm on the speed of 0.04 seconds, we consider another controller design. We take a lower sampling period of 0.1 seconds at which the controller system is still stable and calculate the discrete controller. Figure B.3 shows the step response for this controller system and we see that the settling time is much larger.

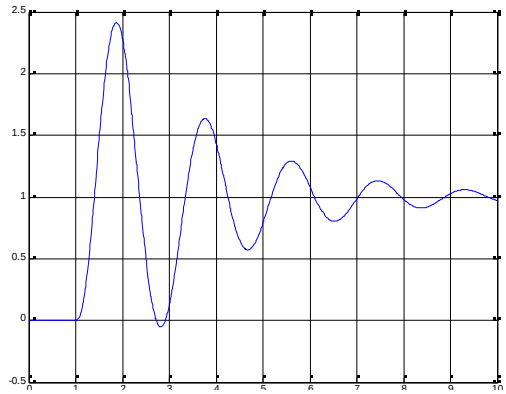

Figure B.3 Step response.

We compute for this system the normalize $SSF_s(s)$ of 1% and is shown as in figure B.4 (red line). The same procedure we have done with different sampling period and are all shown in figure B.4. We see that the controller system becomes more sensitive when a lower sampling period is used.

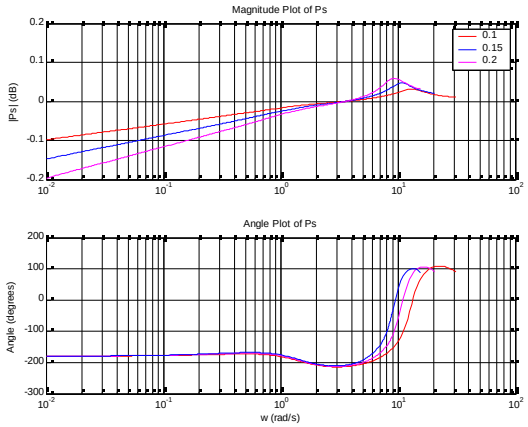

Figure B.2 SSF for different sampling periods.

tf2zT.m

```
% Author:   Maurice Snoeren
% Date:     27/10/2004
% Info:     This function converts a "sys" format to a function based on the
%           symbolic z and T. A conversion type can be set to tustin, forward
%           or backward. Default tustin is used.
%
% syntax:   tf2zT(Hs, conversion)
%
function [HzT] = tf2zT(Hs, conversion)

syms s z T; % Define the symbolic variable name

switch (conversion)
    case 'forward'
        disp('Conversion type is set to forward')
        s = sym('(z-1)/(T)');
    case 'backward'
        disp('Conversion type is set to backward')
        s = sym('(z-1)/(z*T)');
    otherwise
        disp('Conversion type is set to tustin')
        s = sym('(2/T)*((z-1)/(z+1))');
end

[den, num] = tfdata(Hs, 'v'); % Get the data of the transfer function

% Compute the symbolic function
d = size(den,2);
n = size(num,2);
f_den = 0;
f_num = 0;
for i=1:d
    f_den = f_den + den(i)*s^(d-i);
end
for i=1:n
    f_num = f_num + num(i)*s^(d-i);
end

% Return the symbolic function
HzT = simple(f_den/f_num);
```

bodesyms.m

```
% Author:   Maurice Snoeren
% Date:     27/10/2004, updated 12/01/2005
%
% Info:     Generate bode plot of an given symbolic transfer function.
%           When the sampling period is specified the function stops when
%           the frequency becomes (1/Ts)*pi. This is done for sampled-data
%           systems which can also contain the z-transform (exp(st)).
%
% syntax:   bodesyms(Ps, Ts)
%
function [f_w, f_abs, f_angle] = bodesyms(Ps, Ts)

syms s w T; % Define the symbolic variable names

% Substitute s=jw for the bode plot
a    = sym(i*w);
f_jw = subs(f_s,s,a)

% Generate the data
n = 1;
dec_b = -2; % Configuration of the decades we want to calculate
dec_e =  2;
for dec=dec_b:1:dec_e
    for c=0:0.05:8
       w_i = 10^dec + 10^dec*c;
       if (Ts == 0)
         f_jw_subs = subs(f_jw,w,w_i);
         f_abs(n)  = log10(abs(f_jw_subs))*20;
         f_angle(n)= angle(f_jw_subs)*360/(2*pi);
         f_w(n) = w_i;
       else
           if (w_i < ((1/Ts)*pi))
             f_jw_subs = subs(f_jw,w,w_i);
             f_abs(n)  = log10(abs(f_jw_subs))*20;
             f_angle(n)= angle(f_jw_subs)*360/(2*pi);
             f_w(n) = w_i;
           end
       end
       n = n + 1;
   end
end
```

ssf.m

```
% Author:   Maurice Snoeren
% Date:     11/01/2005
%
% Info:
% This function calculates the SSF (sensitivity sampling frequentie)
% function of a given symbolic transfer function.
%
% Definition of the ssf(z,T)=ssf(s,T) and is defined as follows:
% ssf(s,T) = d(H(z,T)|z=e^(sT))/dT = d(H(s,T))/dT
%
% Syntax:
% SSF_Hz = ssf(H_sT)
%
function [SSF_Hz] = ssf(H_sT)

syms z T s w; % Define symbolic variable names

% Calculate the ssf
SSF_Hz = simple(diff(H_sT, T))
```

tf2s.m

```
% Author:   Maurice Snoeren
% Date:     27/10/2004
% Info:     This function converts a "sys" format to a function based on the
%           symbolic s.
%
% syntax:   tf2s(Hs)
%
function [Hs_s] = tf2s (Hs)

syms s; % Define the symbolic variable name

[den, num] = tfdata(Hs, 'v'); % Get the data of the transfer function

% Compute the symbolic function
d = size(den,2);
n = size(num,2);
f_den = 0;
f_num = 0;
for i=1:d
    f_den = f_den + den(i)*s^(d-i);
end
for i=1:n
    f_num = f_num + num(i)*s^(d-i);
end

% Return the symbolic function
Hs_s = simple(f_den/f_num);
```

ssf_plot.m

```
% Author:   Maurice Snoeren
% Date:     11/01/2005
%
% Info:
% This function calculates the SSF of a given symbolic transfer function of a
% process and a controller and plot the frequency response. The transfer can be
% set to T, S, R and input. Ps has to contain the symbolic s. Cz has to contain
% the symbolics z and T. The plot is normalized on n%.
%
function  [SSF] = ssf_plot (Ps, Cz, Ts, transfer, n)

syms s z T; % Define the symbolic variable names

CsT = subs(Cz,z,exp(s*T)); % Substitute the z with exp(sT)

switch (transfer)
    case 'T'
     Hs = ((1/T)*(1-exp(-s*T)/s)*(Ps*CsT))/(1+((1/T)*(1-exp(-s*T)/s)*(Ps*CsT)))
    case 'S'
     Hs = (1)/(1+((1/T)*(1-exp(-s*T)/s)*(Ps*CsT)))
    case 'R'
     Hs = (CsT)/(1+((1/T)*(1-exp(-s*T)/s)*(Ps*CsT)))
    case 'input'
     Hs = (Ps*CsT)/(1+(Ps*CsT)) * (1/(s^2+1))
    otherwise
     error('unknown transfer function type.');
end

SSF   = ssf(Hs); % Get the SSF of the transfer
SSF_T = subs(SSF, T, Ts);
generate_bode_plot(SSF_T, 'b', Ts, n);
end

function generate_bode_plot (H_s, color, SP, n)

syms s z T;
[var_t, var_abs, var_angle] = bodesyms(H_s, SP);
var_abs = var_abs*(SP*n/100); %normalize the result

% Unwrap the angle, because it is distorted
var_angle = var_angle*((2*pi)/360);
var_angle = unwrap(var_angle);
var_angle = var_angle*(360/(2*pi));

figure(1); subplot(2,1,1); % Generate the magnitude plot
semilogx(var_t,var_abs, color);
title('Magnitude Plot of Ps');
ylabel('|Ps| (dB)'); grid on; hold on;

subplot(2,1,2);  % Generate the angle plot
semilogx(var_t,var_angle, color);
title('Angle Plot of Ps');
xlabel('w (rad/s)'); ylabel('Angle (degrees)'); grid on; hold on;

end
```

APPENDIX C

DERIVATION SAMPLED-DATA SYSTEM FREQUENCY RESPONSE CHAPTER 7

We shall derive the sampled-data system shown in figure C.1. This is a standard implementation of a sampled-data system that is used in practice. The output *y(t)* is sampled and the reference *r(k)* is discrete. Two extra loops are added to the system, which represents the transfers that are introduced by the disturbances of the model that is described in chapter 6. Furthermore, standard disturbances are also implemented in this sampled-data system.

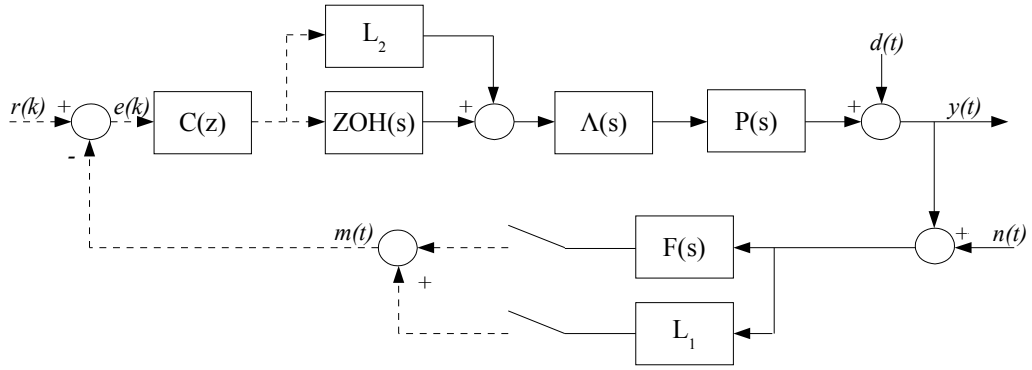

Figure C.1 Sampled-data system model with L_1 and L_2 as the complex loops of the disturbances.

We can write the relations among the Laplace transforms as

$$\begin{aligned} E^*(s) &= R^*(s) - M^*(s), \\ M(s) &= \big(F(s) + L_1(s)\big)\big(Y(s) + N(s)\big), \\ Y(s) &= E^*(s)C^*(s)\big(ZOH(s) + L_2(s)\big)\Lambda(s)P(s) + D(s). \end{aligned} \tag{C.1}$$

Next we sample these equation and use the periodic property, so that we can extract the periodic transfers. The equations become then

$$\begin{aligned} E^*(s) &= R^*(s) - M^*(s), \\ M^*(s) &= \Big(Y(s)\big(F(s) + L_1(s)\big)\Big)^* + \Big(N(s)\big(F(s) + L_1(s)\big)\Big)^*, \\ Y^*(s) &= E^*(s)C^*(s)\Big(\big(ZOH(s) + L_2(s)\big)\Lambda(s)P(s)\Big)^* + D^*(s). \end{aligned} \tag{C.2}$$

We begin with Y^* and substitute E^*, so we get

$$\begin{aligned} Y^*(s) &= R^*(s)C^*(s)\Big(\big(ZOH(s) + L_2(s)\big)\Lambda(s)P(s)\Big)^* \\ &- M^*(s)C^*(s)\Big(\big(ZOH(s) + L_2(s)\big)\Lambda(s)P(s)\Big)^* + D^*(s). \end{aligned} \tag{C.3}$$

When we substitute M^* we get

$$\begin{aligned} Y^*(s) &= R^*(s)C^*(s)\Big(\big(ZOH(s) + L_2(s)\big)\Lambda(s)P(s)\Big)^* \\ &- \Big(Y(s)\big(F(s) + L_1(s)\big)\Big)^* C^*(s)\Big(\big(ZOH(s) + L_2(s)\big)\Lambda(s)P(s)\Big)^* \\ &- \Big(N(s)\big(F(s) + L_1(s)\big)\Big)^* C^*(s)\Big(\big(ZOH(s) + L_2(s)\big)\Lambda(s)P(s)\Big)^* + D^*(s). \end{aligned} \tag{C.4}$$

We see that *Y(s)* is bound up with *(F(s)+L1(s))* and cannot divided out to give a transfer function. However we can use the definition of sampled signals, so the equation becomes

$$\begin{aligned} Y^*(s) = {} & R^*(s)C^*(s)\frac{1}{T}\sum_{n=-\infty}^{\infty}\left(ZOH(s-jnw_s)+L_2(s-jnw_s)\right)\Lambda(s-jnw_s)P(s-jnw_s) \\ & -C^*(s)\frac{1}{T}\sum_{n=-\infty}^{\infty}Y(s-jnw_s)\left(F(s-jnw_s)+L_1(s-jnw_s)\right) \\ & \frac{1}{T}\sum_{n=-\infty}^{\infty}\left(ZOH(s-jnw_s)+L_2(s-jnw_s)\right)\Lambda(s-jnw_s)P(s-jnw_s) \\ & -C^*(s)\frac{1}{T}\sum_{n=-\infty}^{\infty}N(s-jnw_s)\left(F(s-jnw_s)+L_1(s-jnw_s)\right) \\ & \frac{1}{T}\sum_{n=-\infty}^{\infty}\left(ZOH(s-jnw_s)+L_2(s-jnw_s)\right)\Lambda(s-jnw_s)P(s-jnw_s) \\ & +\frac{1}{T}\sum_{n=-\infty}^{\infty}D(s-jnw_s). \end{aligned} \tag{C.5}$$

It is possible to obtain the fundamental transfer function of *Y(s)* by *n=0* and become

$$\begin{aligned} Y_f(s) = {} & R(e^{st})C(e^{st})\frac{1}{T}(ZOH(s)+L_2(s))\Lambda(s)P(s) \\ & -C(e^{st})\frac{1}{T^2}Y(s)\left(F(s)+L_1(s)\right)\left(ZOH(s)+L_2(s)\right)\Lambda(s)P(s) \\ & -C(e^{st})\frac{1}{T^2}N(s)\left(F(s)+L_1(s)\right)\left(ZOH(s)+L_2(s)\right)\Lambda(s)P(s)+D(s). \end{aligned} \tag{C.6}$$

Rearranging the equation and get all variable together we get

$$\begin{aligned} Y_f(s) = {} & \frac{C(e^{st})\frac{1}{T}(ZOH(s)+L_2(s))\Lambda(s)P(s)}{1+C(e^{st})\frac{1}{T^2}\left(F(s)+L_1(s)\right)\left(ZOH(s)+L_2(s)\right)\Lambda(s)P(s)}\left(R(e^{st})-N(s)\right) \\ & +\frac{1}{1+C(e^{st})\frac{1}{T^2}\left(F(s)+L_1(s)\right)\left(ZOH(s)+L_2(s)\right)\Lambda(s)P(s)}D(s). \end{aligned} \tag{C.7}$$

With this fundamental equation we are able to extract also the fundamental sensitivity functions of the sampled-data system. The sensitivity function becomes

$$S_f(s) = \frac{1}{1+C(e^{st})\frac{1}{T^2}\left(F(s)+L_1(s)\right)\left(ZOH(s)+L_2(s)\right)\Lambda(s)P(s)}. \tag{C.8}$$

And the complementary sensitivity function becomes

$$T_f(s) = \frac{C(e^{st})\frac{1}{T}(ZOH(s)+L_2(s))\Lambda(s)P(s)}{1+C(e^{st})\frac{1}{T^2}\left(F(s)+L_1(s)\right)\left(ZOH(s)+L_2(s)\right)\Lambda(s)P(s)}. \tag{C.9}$$

REFERENCES

[1] Årzén, K.E.
A SIMPLE EVENT-BASED PID CONTROLLER.
In: Draft paper submitted to IFAC World Congress 1999

[2] Åstrom, K.J. And B. Wittenmark
COMPUTER CONTROLLER SYSTEMS, 2ND EDITION.
New Jersey, Prentice – Hall, 1990 (1st edition 1984, 3rd edition 1997)

[3] Bekey, G.A. And R. Tomovic
SENSITIVITY OF DISCRETE SYSTEMS TO VARIATION OF SAMPLING INTERVAL.
Automatic Control, IEEE Transactions on volume 11, Issue 2, April, pages 284 – 287, 1966

[4] Braslavsky, J.H.
FREQUENCY DOMAIN ANALYSIS OF SAMPLED-DATA CONTROL SYSTEMS.
PhD Thesis, The Department of Electrical and Computer Engineering, University of Newcastle, Newcastle, Australia, October 1995

[5] Buttazzo, G.C.
HARD REAL-TIME COMPUTING SYSTEM, PREDICTABLE SCHEDULING – ALGORITHMS AND APPLICATIONS.
Springer, 2004, 2nd edition, ISBN 0-387-23137-4

[6] Cervin, A., D. Henriksson, B. Lincoln, J. Eker and K. Årzén
HOW DOES CONTROL TIMING AFFECT PERFORMANCE.
IEEE Control Systems Magazine, Volume 23, Issue 3, June 2003, pages 16 – 30

[7] Cloudt, R.P.M.
ON THE DESIGN OF DIGITAL MOTOR CONTROLLERS WITH LOW SAMPLING RATES.
M. Sc. Thesis, Measurement and Control group, The department of Electrical Engineering, Eindhoven University of Technology, Eindhoven, The Netherlands, Report nr. 04A/07, 2004

[8] Damen, A.A.H.
INTRODUCTION TO DIGITAL CONTROL.
Lecture notes, Measurement and Control group, Department of Electrical Engineering, Eindhoven University of Technology, 2001.

[9] Damen, A.A.H. And S. Weiland
ROBUST CONTROL
Lecture notes, Measurement and Control group, Department of Electrical Engineering, Eindhoven University of Technology, 2002.

[10] Eker, E., J. Stanley and V. Adve
A FEEDBACK SCHEDULER FOR REAL-TIME CONTROLLER TASKS.
IFAC Control Engineering Practice 2000, 8(12), pages 1369 – 1378, December, 2000

[11] Florescu, O., J. Voeten and H. Corporaal
A UNIFIED MODEL FOR ANALYSIS OF REAL-TIME PROPERTIES.
In preliminary Proceedings of the 1st International Symposium on Leveraging Applications of Formal Methods (IsoLa), TR-2004-6, pages 220 – 227, Paphos, Cyprus, 2004

[12] Florescu, O., J. Voeten and J. Huang and H. Corporaal
ERROR ESTIMATION IN MODEL-DRIVEN DEVELOPMENT FOR REAL-TIME SOFTWARE.
In Proceedings of the Forum on specification and Design Languages (FDL), ISSN 1636-9874, pages 228-239, Lille, France, 2004

[13] Franklin, G.F., J.D. Powell and A. Emami-Naeini
FEEDBACK CONTROL OF DYNAMIC SYSTEMS.
Addison-Wesley Publishing Company, Inc., Third edition, 1994, ISBN 0-201-52747-2

[14] Franklin, G.F., J.D. Powell and M. L. Workman
DIGITAL CONTROL OF DYNAMIC SYSTEMS.
Addison-Wesley Publishing Company, Inc., Second edition, 1990, ISBN 0-201-11938-2

[15] Kao, C.Y. And B. Lincoln
SIMPLE STABILITY CRITERIA FOR SYSTEMS WITH TIME-VARYING DELAYS.
Automatica, Volume 40, Issue 8, August 2004, pages 1429 – 1434

[16] Leung, G.M.H., T.P. Perry and B.A. Francis
PERFORMANCE ANALYSIS OF SAMPLED-DATA CONTROL SYSTEMS.
Automatica, Volume 27, No. 4, pages 699 – 704, 1991

[17] Lincoln, B. And A. Cervin
JITTERBUG: A TOOL FOR ANALYSIS OF REALTIME CONTROL PERFORMANCE.
In: Proceedings 41st IEEE Conference on Decision and Control, Volume 2, pages 1319 –

[18] Micheli, G. De, R.K. Gupta
HARDWARE/SOFTWARE CO-DESIGN.
In: Proceedings of the IEEE, Volume 85, Issue 3, March 1997

[19] Nilsson, J.
REAL-TIME CONTROL SYSTEMS WITH DELAYS.
PhD Thesis, Department of Automatic Control, Lund Institute of Technology, Lund, Sweden, 1998

[20] Nilsson, J., B. Bernhardsson and B. Wittenmark
SOME TOPICS IN REAL-TIME CONTROL.
In: Proceedings of the American Control Conference, Philadelphia, Pennsylvania, June 1998

[21] Pitkin, E.T.
AN IMPROVED SAMPLE-AND-HOLD UNIT.
Automatic Control, IEEE Transactions on Volume 16, Issue 5, pages 5126 – 518, October 1971

[22] Schiøler, H., A.p. Ravn and J.D. Nielsen
IMPACT OF SCHEDULING POLICIES ON CONTROLLER SYSTEM PERFORMANCE.
Satelite Event on CoDesign in Real Time Systems (CERTS), Porto, Portugal, July 2003

[23] Soliman, K.K., S.F. Saraya and W.R. El-Wasif
MICROPROCESSOR BASED PID CONTROLLER DESIGN AND IMPLEMENTATION.
Modelling, Measurement and Control, Volume 46, No. 4, 1992

[24] Tomovic R., and G.A. Bekey
ADAPTIVE SAMPLING BASED ON AMPLITUDE SENSITIVITY.
Automatic Control, IEEE Transactions on Volume 11, Issue 2, pages 282 – 284, April 1966

[25] Wittenmark, B., J. Nilsson and M. Törngren
TIMING PROBLEMS IN REAL-TIME CONTROL SYSTEMS.
Proceedings of the 1995 American Control Conference, Volume 3, pages 2000 – 2004, 1995

[26] Yamamoto, Y.
A RETROSPECTIVE VIEW ON SAMPLED-DATA – CONTROL SYSTEMS.
CWI Quarterly, Volume 9 (1996), No. 3, pages 261 – 276