Stability analysis of sampled-data systems with network delays

Hagenaars, H.L.

Award date:
2005

Stability Analysis of Sampled-Data Systems with Network Delays

H.L.Hagenaars

Master's Thesis

Report No. DCT 2005.136

Supervisor: Prof. Dr. H. Nijmeijer
Coaches: Dr. Ir. N. v.d. Wouw
Ir. M.B.G. Cloosterman

Eindhoven University of Technology
Department of Mechanical Engineering
Dynamics and Control Group

Eindhoven, November 14, 2005

Preface

This report, "Stability Analysis of Sampled-Data Systems with Network Delays", covers the master's thesis study of the author, which has been performed within the Dynamics and Control Group of the faculty of Mechanical Engineering at the Eindhoven University of Technology, under the supervision of prof. dr. Henk Nijmeijer. Coaches during the work presented in this thesis were dr. ir. Nathan van de Wouw and ir. Marieke Cloosterman, both members of the Dynamics and Control Group.

The report is accompanied by a CD-ROM, which contains an electronic version of this report, all the code-files listed in the appendix of this report and all files that can be used to conduct experiments with the setup described in this report. Furthermore, a brief manual is included on how to use the hardware and software developed during the research presented in this thesis.

Eindhoven, November 14, 2005

Abstract

As a result of the ongoing development and decreasing costs of embedded computing systems, the application of embedded control systems increases. In such systems, resources are limited and these limitations need to be taken into account during the design of such systems.
Also the use of data networks increased over the years. As a result, data networks also made their entrance in control systems. Several devices in a control system can be attached mutually using network connections, which increases the system flexibility and the ease of maintenance of such systems. This thesis deals with the influence of the use of real-time data networks to close a control loop on the stability of the controlled system. Such systems are referred to as Networked Control Systems (NCSs).

Network transfers can be considered discrete events which makes a networked control system a sampled data system. Network transfers introduce a delay in the control loop, which is known to degrade the performance of a control system and to even affect its stability. The characteristics of this network induced delay are dependent on the type of network. Network induced delays can be time-varying or even non-deterministic.
The choice of the sample-rate plays an important role in an NCS. Whereas in sampled data systems an increasing sample-rate leads to a better performance, increasing the sample rate in an NCS leads to a higher network load which results in increasing delay in the loop. Therefore a sensible choice of the sample-rate is crucial in NCS design.
The choice of the way of sampling of the various devices in an NCS also influence its characteristics. When making use of event-driven devices, no extra delay is introduced in the loop, but the sample-time is not constant which makes it hard to analyze such systems. The use of time-driven devices introduces some extra delay on top of the delay induced by the network but the sample-rate is fixed and the delay in the loop becomes constant. The introduction of a time-skew between sampling instants of separate devices can be helpful to minimize the extra delay introduced by the sampling process.

To analyze the influence of the sample-rate and the network delays on a feedback controlled systems, a discrete-time model is derived of an NCS with a state feedback controller based on several motivated assumptions. Using this model, the stability of the NCS is analyzed using several analytic techniques as wel as numerical simulations. The analysis shows that for a right choice of state feedback, the stability of the closed loop system not necessarily has to suffer from a delay in the control loop. In some cases the controlled system performs better with a certain amount of delay in the loop.

A mobile robot setup has been adapted to make it suitable to conduct experiments for this research and control related experiments in general. Experiment done using this setup proof that the results obtained numerically can be reproduced in practice.

Samenvatting

Door de constante ontwikkeling en dalende kosten van embedded reken-systemen, neemt de toepassing van dit soort hardware in regel-systemen toe. Het gebruik hiervan introduceert echter een aantal beperkingen waarmee rekening dient te worden gehouden tijdens het ontwerp van embedded regel-systemen.
Ook het gebruik van data netwerken is de laatste jaren sterk toegenomen. Dientengevolge hebben data netwerken ook hun intrede gedaan in regel-systemen. De verschillende delen van een regel-systeem kunnen onderling worden verbonden met behulp van een data netwerk, wat de flexibiliteit van het systeem verhoogd en het onderhoud ervan vergemakkelijkt. In dit verslag wordt de invloed onderzocht van het gebruik van een netwerk om de regel-lus te sluiten, op de stabiliteit van het geregelde systeem. Dit soort regel-systemen worden Networked Control Systems (NCS) genoemd.

Netwerk overdrachten kunnen worden gezien als discrete gebeurtenissen en dat maakt een NCS een discreet systeem. Netwerk overdrachten introduceren een tijdvertraging in de regel-lus waarvan algemeen bekend is dat dit de prestaties en de stabiliteit van het geregelde systeem negatief beïnvloedt. De karakteristieken van een tijdvertraging veroorzaakt door een netwerk zijn afhankelijk van het gebruikte netwerk. Deze vertragingen kunnen variëren in de tijd of zelfs niet-deterministisch zijn.
De keuze voor de bemonster-frequentie speelt een belangrijke rol in het ontwerp van een NCS. Terwijl een hogere bemonster frequentie in discrete systemen doorgaans leidt tot een betere prestatie van het geregelde system, leidt dit in een NCS tot een hogere belasting van het netwerk en dat resulteert in grotere tijdvertraging in de regel-lus. Daarom is een verstandige keuze voor de bemonster-frequentie cruciaal in het ontwerp van een NCS.
De keuze voor de methode van bemonsteren van de verschillende systemen binnen een NCS heeft ook invloed op de karakteristieken van een NCS. Bij het gebruik van event-driven systemen wordt er geen extra vertraging in de lus geïntroduceerd, maar de bemonster-frequentie is niet constant en dat maakt de analyse van event-driven systemen ingewikkeld. Het gebruik van time-driven systemen introduceert extra tijdvertraging bovenop de vertraging als gevolg van de netwerk overdracht, maar de bemonster-tijd en de vertraging in de lus zijn constant in dit geval. Het introduceren van een tijdverschuiving tussen het moment van bemonsteren van de verschillende systemen kan ervoor zorgen dat de extra vertraging door het bemonster-proces geminimaliseerd wordt.

Om de invloed van de bemonster frequentie en de tijdvertraging als gevolg van de netwerk overdracht op de stabiliteit van het geregelde systeem te onderzoeken, wordt er een discrete tijd model van een NCS met toestand-terugkoppeling afgeleid onder een aantal gemotiveerde aannamen. Met behulp van dit model wordt de stabiliteit van het geregelde systeem geanalyseerd, gebruik makend van verschillende analytische technieken en numerieke simulaties. Uit

de analyse blijkt dat voor een juiste keuze voor de toestand-terugkoppeling de stabiliteit van het geregelde systeem niet noodzakelijkerwijs negatief wordt beïnvloed. In sommige gevallen presteert het geregelde systeem zelfs beter met een zekere tijdvertraging in de regel-lus.

Tijdens dit onderzoek is een mobiele robot aangepast om deze geschikt te maken voor experimenten ten behoeve van dit onderzoek en regeltechniek gerelateerde experimenten in het algemeen. Experimenten, uitgevoerd met behulp van deze opstelling, bewijzen dat de resultaten behaald met de analyse kunnen worden gereproduceerd in de praktijk.

Contents

Chapter 1

Introduction

Nowadays, the use of computers is widely accepted in everyday life. In this respect, the personal computer, used for numerous different purposes, plays an important role. But, as development of computers goes on and on, the costs of computing systems decrease, whereas the possibilities and reliability increase. This makes the use of computers in domestic appliances, cars and many other (mass-market) products a realistic option. Systems that include a programmable computer but are not a general-purpose computer itself are called *embedded systems*. Today such embedded computing systems by far outnumber desktop computers (see Fig. 1.1). A top-level modern car, for example, contains more than 50 embedded processors performing tasks of varying complexity. Embedded control systems constitute an important sub-class of these embedded computing systems.

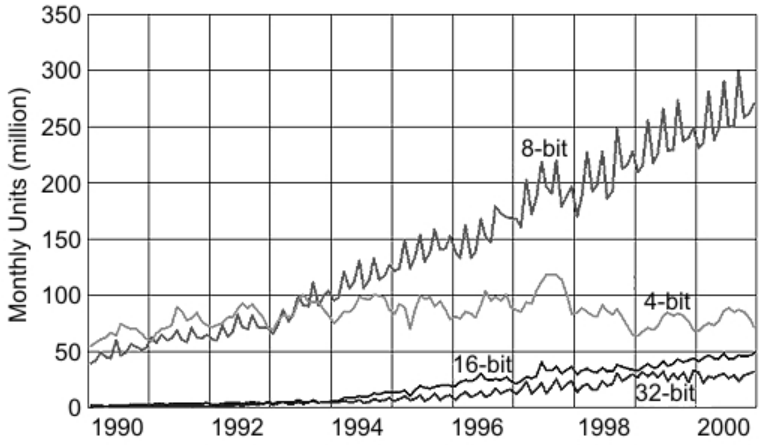


Figure 1.1: World-wide microprocessor sales of all types and all markets (source: www.extremetech.com)

Because of the application of embedded control systems in mass-market products, the design and production of such products is subject to hard economic constraints. Because of these eco-

nomic constraints, low cost and general purpose computing components and other off-the-shelf hardware are preferred above high capacity processors or specially designed components. For the control engineer, these economic constraints give rise to constrained computing speed, limited memory size and limited communication bandwidth. Because of these constraints, sample-time and digital word-length are limited in these embedded control systems. To effectively design such embedded control systems, the consequences of a digital implementation using resource constrained hardware have to be taken into account in the design process beforehand. The term *application aware design* is often used in this respect.

With the increase of computer use, also the use of data networks increases. The first data network protocols evolved over the last 30 to 40 years into modern network protocols such as ethernet and TCP/IP, which are mainly used in general purpose communication networks nowadays. Due to the continuous development and standardization of network communication, the use of network connections also made its entrance in control systems. In stead of hard-wiring all parts of a control system together, networks are used to connect different parts of a control system. For this purpose specialized industrial network protocols such as Profibus, DeviceNet and CAN (Controller Area Network) were developed.
Connecting different parts of a control system using a real-time communication network has several advantages. Because of the accepted standards, communication between different parts or systems of different manufacturers is much easier which makes a modular construction of a control system possible. This increases the system flexibility, and as a result the ease of maintenance, to a large extend. Furthermore existing infrastructures can be used to control processes over a larger distance.
Unfortunately the introduction of a communication network in a control system also imposes extra constraints on the system. One of the major drawbacks of using a network to close the control loop is the introduction of delay in the control loop or even data loss due to network transmissions. Furthermore, the delay induced by the network is strongly dependant on the utilization of the network and therefore this delay is not deterministic in most cases, which makes it very hard to guarantee a certain level of performance of the closed loop system. Even the stability of the system can be affected, which can do serious damage to the system and its environment.

As a result of the developments outlined in the foregoing, analysis and control of so called *Networked Control Systems* is a field of research with increasing interest over the last ten to fifteen years [5]. In this field of research mainly two different approaches can be distinguished. The first approach is from a computer science point of view. The research in this field focusses on development of network protocols and scheduling methods for network traffic to reduce the negative effects of network transfers (see e.g. [6, 12, 28, 29]).
The second approach is more from a control engineering perspective. The network with its unfavorable characteristics is treated as a given fact and the aim is to develop control methodologies that are able to effectively cope with these characteristics.
The research covered in this report adheres to the control engineering point of view.

1.1 Research goals

The purpose of the research dealt with in this thesis is to uncover the problems that arise in the control of networked systems and give a theoretical as well as an experimental analysis of the influence of network parameters on the performance and stability of the controlled system. The goal of this research is summarized in the following main research question:

> *What are the effects of the use of a communication network to close a control loop on the stability of the closed loop system and how can these effects be taken into account in the design process of a control system with a communication network?*

To answer this main research question, the following questions have to be answered respectively:

- Which problems arise in the control of networked systems and what parameters play a role when a communication network is integrated into a control system?
- What is the influence of these parameters on the stability of the controlled system?
- Can the influence of several parameters on the stability of a networked control system be tested on a suitable experimental setup?
- Can one formulate relevant design rules that can serve as a guideline in designing a networked control system?

In order to answers these research questions, the following approach has been followed. A thorough study of the available literature on networked control systems has been performed to identify which parameters play a role and which issues arise in the control of networked systems. With this information, a mathematical model has been derived to analyze the influence of those parameters on the stability of a networked control system, under motivated assumptions, both analytically and using simulations. The validity of these results has been tested by means of experiments on a specially designed setup. This leads to a thorough understanding of the consequences of the use of communication networks in a control system.

1.2 Thesis outline

This thesis is outlined as follows. In chapter 2 an overview is given of the problems arising in the control of networked systems and the parameters that play an important role in this respect. Furthermore, a short summary of the research done in the field of networked control systems and the focus of this research is given. In chapter 3 this information is used to derive a discrete-time model of a networked control system in which the relevant parameters, found in chapter 2, appear. The second part of this chapter presents the result of an analytic analysis of the influence of the used parameters on the stability of the system. Chapter 4 describes the experimental setup that has been created during this research to validate the analytic results experimentally. The identification of this setup and the experimental results are discussed in chapter 5. This thesis will end with a concluding discussion on the design of networked control systems and some recommendations for future research.

Chapter 2

Networked Control Systems

A Networked Control System (NCS) is a control system in which at least a part of the control loop is closed over a communication network. Connecting different parts of a control system using a network has several advantages. Because of the modularity of such a setup, the system is easier to maintain. Furthermore, because of the widespread use of data networks, standard protocols (e.g. ethernet) can be used and a lot of relatively cheap commercial off-the-shelf hardware is available.
Unfortunately, the use of a communication network in a control systems introduces some negative effects. In this chapter, these negative effects will be discussed. Two different purposes for which a network is used in a control system will be discussed in section 2.1. Next, a brief explanation of the two most common types of communication networks is given in section 2.2 to provide insight in the consequences of the use of such networks in a control system. Sample-time selection and the way of sampling both play an important role in the design of an NCS. Several sampling issues will therefore be discussed in section 2.3 of this chapter. This chapter ends with a short summary of the research done in the field of NCSs from a control engineering point of view and the focus of this research.

2.1 General configuration of an NCS

The major system components of a control system are a process to be controlled, a controller and one or several sensors and actuators. Usually these components are hard-wired together. In an NCS however, one or more of these components are connected using a communication network. This network can also be shared with other control loops and other network resources.
Mainly there are two different purposes of the use of a network in a control system. Consequently, two configurations can be distinguished; the *direct structure* and the *hierarchical structure* [27].
In the direct structure (Fig. 2.1), a remote system consisting of a physical plant with sensors and actuators is connected to a main controller using a network. The control loop is closed over the network. Sensor signals are sent to the controller via the network connection. The controller will send the computed control signals to the actuator using either the same network line or a different line. This configuration is used in situations where the computing

hardware is physically separated from the process to be controlled or where one control unit controls multiple processes.

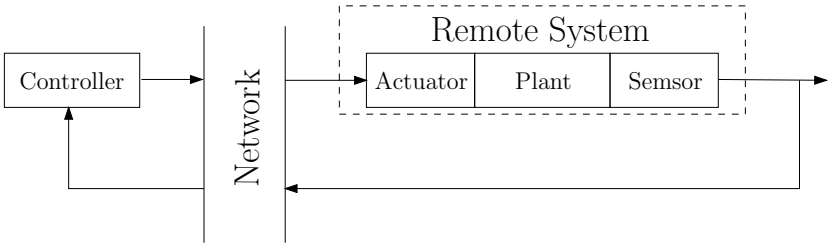


Figure 2.1: NCS with the direct structure.

In the hierarchical structure (Fig. 2.2), the remote system consists of a plant that is already controlled in closed loop by a controller that is part of the remote system. A main controller is connected to the remote system using a network. Typically the sample rate in the remote control system is higher than the sample rate of the networked control loop. The remote controller is used to accurately control the remote process. The main controller can then be used to perform more sophisticated tasks or even to connect different remote control systems. This type of control is often referred to as supervisory control.

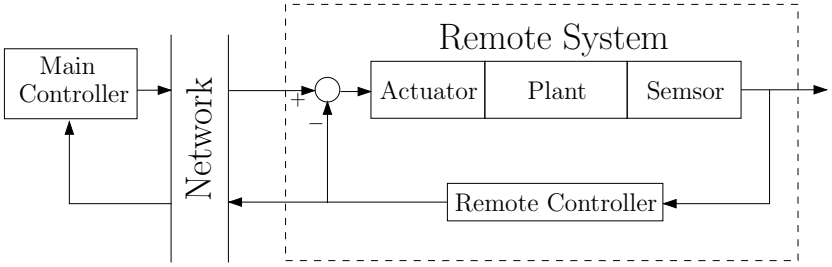


Figure 2.2: NCS with the hierarchical structure.

Which configuration is most suitable depends on the type of application and its requirements. Research in the field of NCSs mainly focusses on the direct structure. The results however can be applied to the hierarchical structure as well, because the remote control system can be treated as a pure plant. Obviously many variants of the two basic configurations shown in Fig. 2.1 and Fig. 2.2 exist.
The type of network used is also a key issue in the NCS configuration. Different types of networks exist that each have their specific use and therefore specific characteristics. These characteristics will be explained in more detail in the next section.

2.2 Shared communication networks

In order to understand what the influence of the use of a network in a control system is, some knowledge of how a network works and what different types of networks can be used, is required. All communication networks serve the same purpose; namely to transport information between devices on the network. The main idea behind the network traffic is the same.

There are some differences however in the way network traffic is controlled. Due to these differences, the characteristics of a network in an NCS depend on the type of network used.

2.2.1 Network traffic

Fig. 2.3 shows a part of a network with four devices attached to it. In the figure these devices are computers, but as mentioned before also many other different devices can be attached to the network. The network serves as a communication medium for all of the devices connected to it. The devices have to be equipped with special network interface hard- and software called a network interface (NI). The NI manages the communications between devices on the network.

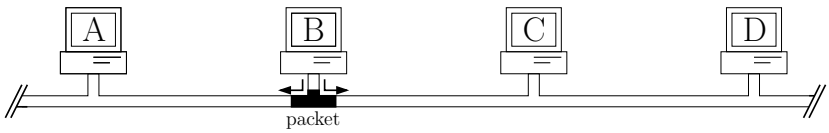


Figure 2.3: Send packages over a network.

When one station wishes to send a message to another device over a shared communication network, it uses the NI to split the message into packets. These packets consist of message data surrounded by a header and trailer that carry special information. The header contains, among other things, the address of the destination device.

When the NI transmits a packet, a stream of data bits, represented by changes in electrical signals, travels along the shared cable. All of the devices attached to it see the packet as it passes by. The NI of each device checks the destination address in the packet header to determine wether the packet is addressed to it. When the packet passes the device it is addressed to, the NI of that device copies the packet and then extracts the data from it so the device can process these data.

Since each individual packet is small, it takes very little time to travel to the ends of the cable where the electrical signal dissipates. So after a packet, carrying a message between one pair of devices, passes along the cable, another device can transmit a packet to whatever device it needs to send a message. In this way, many devices can share the same medium.
When several entities share the same communication medium, some mechanism, called an access method, must be in place to control access to the network in a fair manner.

2.2.2 Random access and token passing networks

There are several access methods which all use a certain amount of the communication bandwidth for the access control. Two of them are mostly used in general purpose communication networks as well as in industrial networks. Based on these access methods, all communication networks can roughly be divided into two main categories, *token passing networks* and *random access networks*.

In a random access network, any device can transmit whenever it needs to send information. To avoid data collisions, specific random access protocols are developed requiring the device to check the network line for availability before transmitting information. This process is called

carrier sensing and therefore random access networks are also called Carrier Sense Multiple Access (CSMA) networks. Even though each device on the network checks the network line before it attempts to transmit, it remains possible for two transmissions to overlap on the network. This overlap is called a collision. To overcome the problem of collision, collision detection has been developed. When a NI detects a collision, it will try to resend the packet after a random pause. Therefore, access to a random access network is unpredictable.

The benefits of random access networks are:

- The access method used in random access networks, does not increase the network load. No extra data have to be sent over the network to control the network access.
- Data throughput is high at low traffic rates (i.e. limited number of network transitions per time-unit) because the number of collisions is minimal in this case.

The disadvantages of random access networks are:

- At high traffic rates, data collisions and the resulting retransmissions diminish performance dramatically. It is theoretically possible that collisions can be so frequent at higher traffic levels that no device has a chance to transmit.
- Due to the retransmissions and the time it takes to sense collisions a delay is introduced. This delay is non-deterministic because of the used access method.

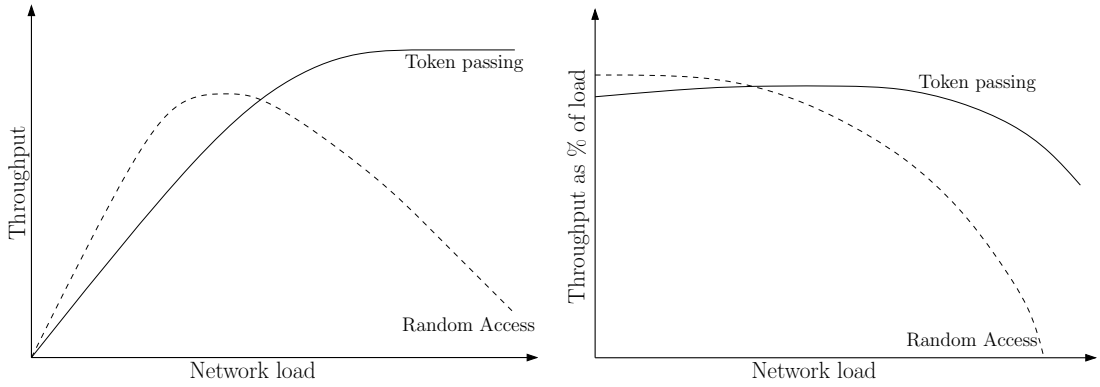
Token passing networks use a different access method. A special authorizing packet is used to inform devices that they can transmit data. This packet is called a token and it is passed around the network in an orderly fashion from one device to the next. Devices can transmit only if they have control of the token. This method distributes the access control among all the devices evenly.
Token passing provides the following advantages:

- Token passing offers the highest data throughput possible under high traffic conditions. Only one transmission can occur at a time and collisions cannot occur. Therefore, token passing experiences less performance degradation at higher traffic levels.
- Token passing is deterministic. Each station is guaranteed an opportunity to transmit each time the token travels around the network. Therefore the delay introduced due to network transmissions is deterministic in token passing networks.
- As the traffic increases, data throughput also increases to a certain level and then stabilizes.

The disadvantages of token passing are as follows:

- Token passing protocols introduce a higher bandwidth overhead then random access networks because extra network traffic is necessary to pass around the token.
- All devices require complicated software that needs to be modified whenever a device is added or removed.

Random access and token passing networks have different performance characteristics. Fig 2.4(a) shows the network throughput as a function of the network load for both network types. The throughput of a random access network rises smoothly with increased traffic levels up to a certain level. At that level, collisions begin to occur with greater frequency, resulting in a gradual reduction in network throughput. At some point, the network throughput reaches an unacceptably low level.
Token passing exhibits lower throughput at lower traffic levels. This is a result of the communication bandwidth that has to be used for token passing. The throughput rises smoothly until the network is fully utilized. At that point, the throughput stabilizes. The throughput does not degrade because no collisions can occur. However, at this network load, all workstations are sharing a strictly limited communication bandwidth. Although the total throughput remains stable, the communication bandwidth available to a given station diminishes as demand increases.
In Fig 2.4(b), the throughput is given as a percentage of the load which gives a better view of the performance of the network. It illustrates how throughput decreases as a percentage of demand. Basically, as demand increases, a smaller percentage of the demand can be satisfied. With random access networks, the fall-off after a certain point is fairly rapid until the number of collisions interferes with virtually all traffic on the network and almost no packets are actually delivered.
The performance of a token passing network also declines, but never reaches zero. Each device is guaranteed a deterministic share of the network's bandwidth, although this share may, at some point, be considered inadequate for the applications that make use of the network.



(a) Absolute network throughput (b) Network throughput as perceived by a user

Figure 2.4: Absolute network throughput and as a percentage of the network load.

2.2.3 Network induced delays

Due to the time it takes to code data into network packets and the used access method, a network transmission introduces a delay. For random access networks this delay is clearly non-deterministic because of the unpredictable occurrence of collisions and the way these collisions are handled in random access networks. For token passing networks the delay can be computed for a given network utilization and number of devices that share the network. Furthermore, these delays increase as network utilization increases and even data-loss can be

a consequence.
It is well known from control theory that delays can degrade the performance of a control system considerably and they can even make a system unstable. The delays induced by a network transfer can be quite large compared to e.g. computational delays. Therefore it is important to take these network induced delays into account in the design and analysis of NCSs. Designing a controller for a networked system under the assumption that no delays are present would lead to poor results or even to an unstable control system.

2.3 Sampling issues in NCSs

The network transmissions can be regarded as discrete events which makes an NCS a sampled-data system. Furthermore, when one wants to take the effects of a digital implementation into account in the design process, a discrete-time approach is preferred over a continuous-time approach. In the design of a digital control system, issues concerning sampling arise. Because of the limited communication bandwidth of data networks, sampling issues play an even more important role in NCSs. In this section these sampling issues will be discussed.

2.3.1 Sample-rate selection

The performance of sampled-data systems in general is strongly dependant on the sampling time. Normally a sampling rate can be chosen depending on the desired bandwidth but within the limitations of the used hardware. For the networked case however, there is another limiting factor. As sampling periods get smaller, the network traffic load becomes heavier and long time delays result as explained in the previous section [15]. These time delays degrade the performance of the controlled system. In Fig. 2.5 this is depicted graphically.

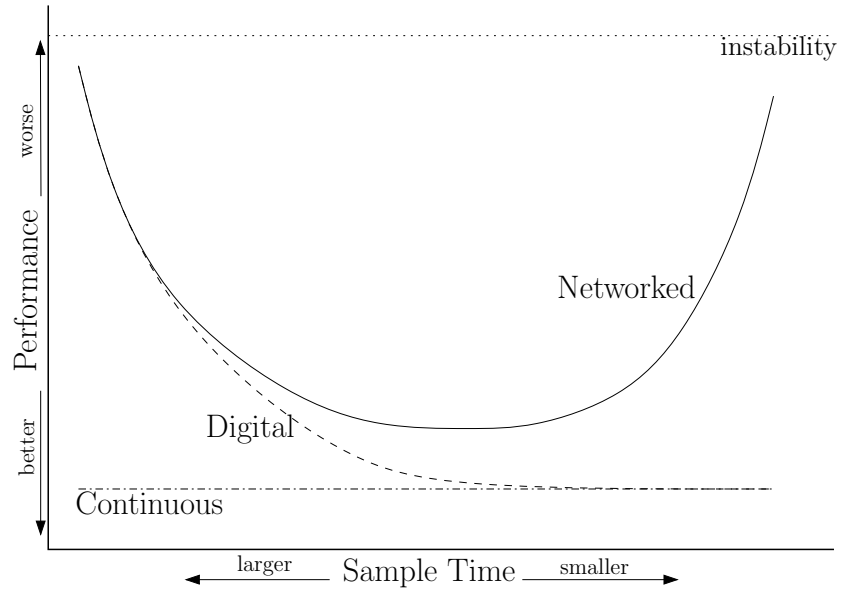


Figure 2.5: Comparison of performance in continuous, digital and networked control systems.

Because the use of a network imposes a limit on the sample-rate, the selection of the sample rate is a very important issue. Depending on the characteristics of the network, the length of the data samples that have to be sent over the network and the number of devices transmitting over the network an optimal sample rate exists. Due to the time-varying nature of the delay or even the non-determinism of the delay in some cases, it is not easy to find this optimal sample rate.

2.3.2 Event-driven vs. time-driven

Sensors, controllers and actuators can operate either in an event-driven or a time-driven fashion. In a time-driven device, input reception or output transmission is controlled by a fixed sample-time, which can be set using a clock signal. An event-driven device is triggered by its input and starts processing immediately when the device receives an input.

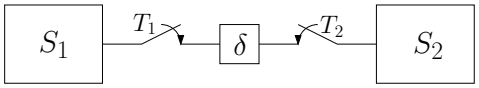
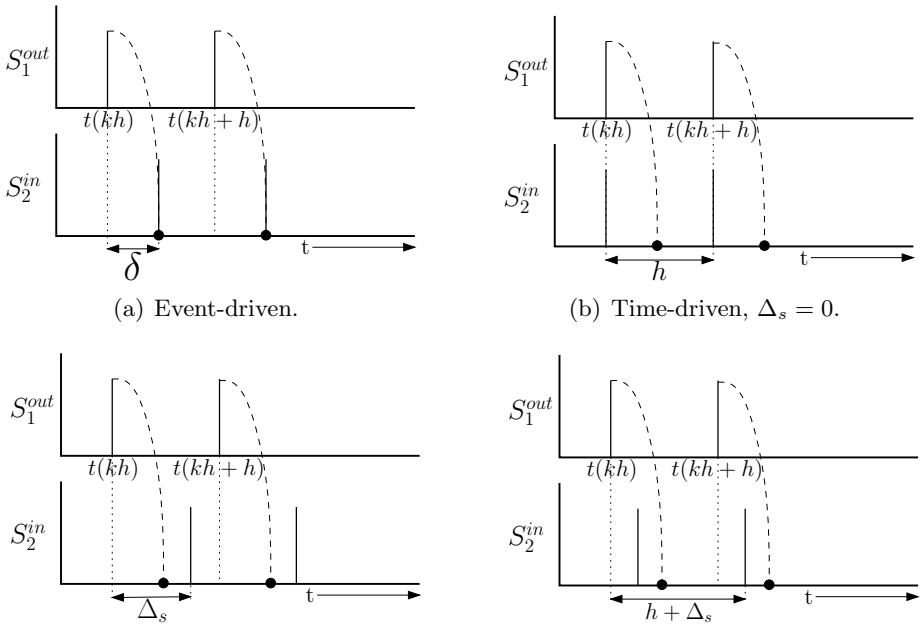


Figure 2.6: Sensor and actuator connected using a communication channel with a constant delay δ.

Consider for example Fig. 2.6. Two systems S_1 and S_2 are connected using a communication channel which introduces a delay of δ in the transfer of data between the two systems. Suppose system S_1 is a sensor that sends out data samples with a fixed sample-time $T_1 = h$ to system S_2 that can be regarded as an actuator that processes the data received from S_1. Also assume that the delay $\delta < h$.
Suppose S_2 is an event-driven device, i.e. the data sent by S_1 will be processed as soon as they arrive at S_2.
Fig. 2.7(a) shows the timing of the signals in this case. The upper time-line shows the discrete sampling instants of the sensor S_1 at time instants $t(kh)$ and $t(kh + h)$. The data arrive at S_2 after a delay δ, where they will be processed immediately. The effective delay between S_1 and S_2 is δ in this case.
Suppose S_2 is a time-driven actuator, i.e. S_2 samples its input with a fixed sample-rate T_2. Under the assumption that $T_1 = T_2 = h$, three different cases can be distinguished.
In the first case, which is illustrated in Fig. 2.7(b), the sampling instant of S_1 is exactly the same as the sampling instant of S_2. At $t(kh)$ sensor S_1 sends a sample to S_2. This sample is available at S_2 at time instant $t(kh + \delta)$ but S_2 starts to process the sample at $t(kh + h)$. The effective delay between S_1 and S_2 is h in this case.
The second and third possible case are illustrated in Figs. 2.7(c) and 2.7(d) respectively. In both cases the sampling instant of S_2 is shifted a time Δ_s compared to the sampling instant of S_1. Fig. 2.7(c) illustrates the case where $\Delta_s > \delta$. S_2 can process the sample sent by S_1 at $t(kh)$ at time $t(kh + \Delta_s)$. The effective delay between S_1 and S_2 is in this case the time-skew Δ_s, which is by definition always smaller than h.
If $\Delta_s < \delta$ however, the sample sent by S_1 at $t(kh)$ will be processed by S_2 at $t(kh + h + \Delta_s)$ as illustrated in Fig. 2.7(d). This introduces an effective delay of $h + \Delta_s$, which is the worst of all cases.



(a) Event-driven.

(b) Time-driven, $\Delta_s = 0$.

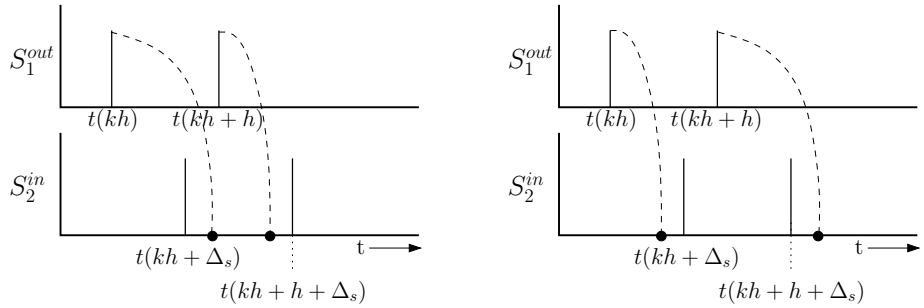
(c) Time-driven, $\Delta_s > \delta$.

(d) Time-driven, $\Delta_s < \delta$.

Figure 2.7: Event-driven versus time-driven operation of actuators.

If the delay δ is not constant but a random variable, the two last cases can occur successively. Suppose the sample sent to S_2 at $t(kh)$ is subject to a delay of $\delta > \Delta_s$ and the sample sent to S_2 at $t(kh+h)$ is subject to a delay $\delta < \Delta_s$. This situation is depicted in Fig. 2.8(a). The first sample arrives at S_2 after its sampling instant. The second sample arrives before its next sampling instant. Two data samples arrive at S_2 during the same sampling interval, so the first sample will be rejected. This phenomenon is referred to as *data rejection*.
The opposite case can also occur. Suppose the sample sent to S_2 at $t(kh)$ is subject to a delay of $\delta < \Delta_s$ and the sample sent to S_2 at $t(kh+h)$ experiences a delay $\delta > \Delta_s$. In the sampling interval of S_2, between $t(kh+\Delta_s)$ and $t(kh+h+\Delta_s)$, no data arrive at the controller. In this case the previous sample will be processed twice as illustrated in Fig. 2.8(b). This phenomenon is referred to as *vacant sampling* [9, 10].
Note that when there is no time-skew between the sampling instants of S_1 and S_2, vacant sampling and data rejection can also occur if the delay δ can get larger than the sample-period h occasionally.

When the actuator is event-driven, the delay between the sensor and the actuator is exactly the delay induced by the network. However, this delay is time-varying or even non-deterministic. This makes the sample time of the actuator time-varying or non-deterministic which makes it very hard to analyze such a configuration. Control strategies can be developed that compute a control signal based on the delay of the current sample. For this type of approach time-stamping is used. The time of generation is attached to every sample. Based on this information the controller can determine how old the sample is and use this information in the computation of the control signal. The main drawback of this approach is that this time-stamp has to be sent, which introduces extra delay that degrades the performance. Furthermore the clocks of both the controller and the remote system have to be synchronized in order to compute the right value of the delay. Clock synchronization however, is not only



(a) Data rejection. (b) Vacant sampling.

Figure 2.8: Illustrations of vacant sampling and data rejection.

a problem in event-driven control strategies. It is a general problem in distributed systems. The problem of clock synchronization will be discussed in the next sub-section.

In the time-driven case the effective delay is always larger than the delay induced by the network. For a proper choice of the sample-rate and the time-skew Δ_s however, the effective delay is constant which makes it much more easy to analyze such a setup. A sample-rate can be chosen that gives an optimal throughput and consequently minimal delay. If an upper bound of the delay is known, an appropriate time-skew can be chosen to minimize the effective delay in the system and prevent vacant samples or rejected data. To assure a certain time-skew Δ_s the clocks of both the controller and the remote system have to be synchronized as well.

2.3.3 Clock synchronization

In distributed systems, different processors communicate by exchanging data over a network. An NCS is an example of such a distributed system. Consider for example a setup like the direct structure depicted in Fig. 2.1. Suppose the sensor of the remote system samples with a fixed sample-rate. This sample rate is driven by a clock on board the remote system. The controller samples with a fixed sample-rate driven by a clock on board the controller. If both clocks are accurate, it will be no problem to run both the sensor and the controller using the same sample rate. If one wants to guarantee a certain time-skew between the sample instant of the sensor and the controller, the clocks of both devices have to be the same.
Assuming T_S is the local time on-board the sensor device and T_C the local time on-board the controller, then

$$T_S = T_C + \xi,$$

where ξ is the clock mismatch between both devices. If ξ can be measured before the control system is started, both clocks can be synchronized. Under the assumption that both clocks are infinitely accurate, the clocks will always run the same time.
Unfortunately infinitely accurate clocks do not exist and a clock will drift over a certain period of time. Therefore each period, depending on the time-scales of the control system and on the actual clock drift, a re-synchronization has to be performed to keep an acceptable clock synchronization.

Clocks can be synchronized using synchronization algorithms which send time information back and forth between devices to compensate for the clock mismatch. Hardware synchronization can also be used. Special wiring can be used to connect all the devices that need to run on the same clock to a global clock signal. This solution however is rather expensive and clears away the advantages of the use of networks.
Clock synchronization is a research area itself, and it will not be discussed in greater detail in this report.

2.4 Focus of this research

As mentioned before in the introduction this research will focuss on the control engineering view on NCSs. From this control engineering point of view several research contributions can be found in literature. This research can roughly be divided into three separate categories:

- **Modeling of NCSs**
  The analysis of a networked control systems starts with an NCS model. Several different approaches have been taken to model an NCS based on the different parameters of interest. Mainly the discrete-time delayed system formulation is used [3, 8] but also hybrid formulations [32] have been used. Based on the delay assumptions different models have been derived varying from models for constant delays from less than a sampling period up to several sampling periods and periodic delays [9, 10]. Also random network delays have been modeled using the Poisson process and fluid flow models [27] and delay models based on an underlying Markov chain [21] among other approaches. Furthermore, attempts have been made to include packet loss into models [32].

- **Analysis of NCSs**
  Performance and stability analysis plays an important role in NCS research. As explained in this chapter, the delays introduced by a network can degrade the performance of an NCS and even can cause instability. Also sampling issues play an important role as is pointed out in this chapter. Therefore, issues like sample rate selection, sampling methods and network delay characteristics are addressed in literature [7, 15].
  Stability analysis of NCSs also plays an important role. Analytic stability analysis using Lyapunov stability theory and other techniques such as the Jury test have been used to analyze stability of NCSs under different assumptions, using different models and different control techniques [4, 9].

- **Control of NCSs**
  Another focus within the NCS research is the design of control strategies that can compensate for the network induced delays. The most common method used is LQG [15, 21] optimal control, but also state feedback control using a state observer [20, 32], fuzzy logic control, robust control [27] and control techniques for hybrid systems [32] have been proposed . There is little experimental work performed to validate obtained results.

The goal of this research is to uncover the issues involved in the design of NCSs. No complicated control strategies are considered. Whereas a lot of work done in the field of NCS research is based on more sophisticated control strategies, the goal in this thesis is not to

develop control strategies that can compensate for the negative effects introduced by a network, but to analyze the characteristics of an NCS with a standard state feedback controller. The reason for this is that standard control strategies such as state feedback can easily be implemented in an embedded setting. Furthermore the majority of all industrial controllers consist of the standard three term PID controller because it is simple to implement, reliable and adequate for the majority of control problems. Moreover the use of standard state feedback control gives some surprising results.
Another key issue in the research covered in this thesis is experimental validation of the obtained results. Apart from the fact that it is valuable if numerically obtained results can be reproduced in practice, experimental work also uncovers issues that arise in the implementation.

Chapter 3

Modeling and stability analysis of an NCS

In this chapter, the stability of an NCS will be analyzed analytically. For this purpose a discrete time NCS model will be derived that includes several relevant parameters that were identified in the previous chapter such as sample-time and network delays. This discrete-time NCS model is used to analyze an NCS with one-dimensional plant dynamics. Time-domain as well as frequency-domain analysis is performed to obtain a thorough understanding of the phenomena that arise when the one-dimensional plant is controlled with a linear output feedback controller. Next, simulations with a second-order plant will show that the observed phenomena apply to the two-dimensional plant as well. At the end of this chapter a brief discussion about the obtained results is included.

3.1 A Networked control system model

In this section a discrete-time NCS model will be derived. First, the studied configuration is discussed. Next, a few assumptions on which the model is based are stated. At the end of this section, the discrete-time NCS model is derived that is used in the remainder of this chapter to analyze the performance and stability of a plant controlled over a network.

3.1.1 Modeled NCS configuration

Consider the NCS schematic, given in Fig. 3.1. A continuous-time plant **H** is controlled by a controller **C**. The plant and the controller are connected using some kind of data network. Note that the controller can be connected to the input and the output of **H** using the same network line or a different network line. The continuous-time input of **H** is obtained via the zero order hold D/A conversion of the discrete-time control signal computed by the controller. $y(kh)$ is the sampled continuous-time output $y(t)$ with a constant sample-time h
The samples $y(kh)$ are sent to the controller over the network which induces a sensor to controller delay δ_{sc} as explained in the previous chapter.
To control the plant **H**, a discrete-time output feedback controller $u(kh) = -Ky(kh - \delta_{sc})$ will be used. $y(kh - \delta_{sc})$ is the discrete plant output received over the network. $u(kh - \delta_{sc})$

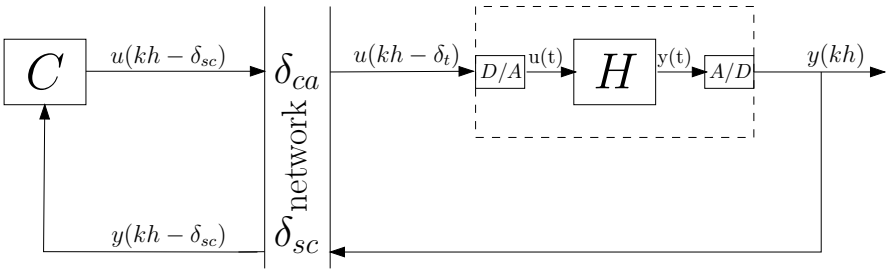


Figure 3.1: Networked Control System schematic.

is the discrete output of the controller during the kth sampling interval. K is a constant gain matrix of appropriate dimensions.
The computed control signal is sent to the actuator over the network which induces a controller to actuator delay δ_{ca}. The discrete control input $u(kh - \delta_t)$ is the discrete control signal received by the plant over the network.

3.1.2 Model assumptions

For the derivation of the NCS model, the following additional assumptions are made.

Assumption 1 *The total network induced delay $\delta_t = \delta_{sc} + \delta_{ca} < h$.*

High sample rates increase network load and therefore increase the network induced delays. Therefore it is sensible to choose a sample-time such that the sample interval is larger than the total network induced delay, i.e. $h > \delta_t$. Deriving a model for other assumptions concerning the total network induced delay δ_t however, is straightforward.

Assumption 2 *The network induced delays δ_{sc} and δ_{ca} are constant.*

As explained in the previous chapter, this can always be obtained when a time-skew is introduced between the sampling instant of the sensor and the actuator that is larger than the upper-bound of the delay. Consequently, no assumption on the time-driven or the event-driven operation of the controller and the actuator has to be made. Note however, that if a time-driven actuator is considered, the moment of actuation is not necessarily the same as the sampling instant of the sensor.

Assumption 3 *There is no computational delay in the controller.*

The computational delay of the controller is not taken into account as a separate parameter. Mainly because the computational delay of a controller is typically much smaller than the network induced delays. Furthermore, if one wishes to take the computational delay of the controller into account, it can be included in δ_{ca} [21].

Assumption 4 *Data loss due to network transfers does not occur.*

As explained in the previous chapter this is not always the case. As network traffic increases data-loss will eventually occur. By choosing an appropriate sample-time, data loss can be minimized or even prevented under strict conditions.

3.1.3 The discrete-time NCS model

The system to be controlled **H** in Fig. 3.1 is a continuous-time linear time-invariant system of which the dynamics is given by:

$$\begin{aligned} \dot{x}(t) &= \mathbf{A}x(t) + \mathbf{B}u(t) \\ y(t) &= \mathbf{C}x(t), \end{aligned} \tag{3.1}$$

where $x(t)$ is the continuous-time state of the system, $u(t)$ is the continuous-time control input, $y(t)$ is the continuous-time output and **A**, **B** and **C** are system matrices of appropriate dimensions.
The discrete-time model of the system given in (3.1), a standard result from digital control theory [8], is given by:

$$\begin{aligned} x(kh+h) &= e^{\mathbf{A}h}x(kh) + \int_{kh}^{kh+h} e^{\mathbf{A}(kh+h-\tau)}\mathbf{B}u(\tau)d\tau \\ y(kh) &= \mathbf{C}x(kh), \end{aligned} \tag{3.2}$$

where h is the used sampling interval.
In (3.2), $y(kh)$ is the sampled output at the sampling instant $t = kh$. $u(\tau)$ is the continuous-time history of the input of **H** over the sampling-interval $[kh \quad kh+h]$. The continuous time input over the sampling interval $u(\tau)$ is piecewise constant due to the zero order hold D/A conversion of the delayed discrete control input $u(kh-\delta_t)$. Note that δ_t is the total network induced delay $\delta_{sc} + \delta_{ca}$.
Under assumptions stated in the previous sub-section, the discrete control input that arrives at the actuator is given by:

$$u(kh-\delta_t) = -Ky(kh-\delta_t). \tag{3.3}$$

Under assumption 1, the piecewise continuous control input $u(\tau)$ has two different values over the sampling interval as illustrated in Fig. 3.2.

$$\begin{cases} u(\tau) = -Ky(kh-h) & \text{for} \quad kh < \tau < kh+\delta_t \\ u(\tau) = -Ky(kh) & \text{for} \quad kh+\delta_t < \tau < kh+h, \end{cases} \tag{3.4}$$

Accordingly, (3.2) can be rewritten as follows [8]:

$$\begin{aligned} x(kh+h) &= e^{\mathbf{A}h}x(kh) - \int_{kh}^{kh+\delta_t} e^{\mathbf{A}(kh+h-\tau)}d\tau\mathbf{B}K\mathbf{C}x(kh-h) \\ &\quad - \int_{kh+\delta_t}^{kh+h} e^{\mathbf{A}(kh+h-\tau)}d\tau\mathbf{B}K\mathbf{C}x(kh). \\ y(kh) &= \mathbf{C}x(kh) \end{aligned} \tag{3.5}$$

When the variables in the integral are changed according to $\eta = kh+h-\tau$ the following form is obtained:

$$\begin{aligned} x(kh+h) &= \Phi x(kh) - \Lambda_0 x(kh) - \Lambda_1 x(kh-h). \\ y(kh) &= \mathbf{C}x(kh) \end{aligned} \tag{3.6}$$

Where $\Phi = e^{Ah}$, $\Lambda_0 = \int_0^{h-\delta_t} e^{A\eta} d\eta BKC$ and $\Lambda_1 = \int_{h-\delta_t}^{h} e^{A\eta} d\eta BKC$.
The closed loop dynamics of an NCS with output feedback as depicted in Fig. 3.1 under the given assumptions can be formulated in state-space notation as:

$$z(k+1) = \Psi z(k) \tag{3.7}$$

Where

$$z = \begin{bmatrix} x(kh) \\ x(kh-h) \end{bmatrix}, \Psi = \begin{bmatrix} \Phi - \Lambda_0 & -\Lambda_1 \\ I & 0 \end{bmatrix}$$

In the closed-loop system matrix Ψ, the controller gain matrix K, the sampling time h and the total network induced delay δ_t are present as parameters. Using this closed loop system matrix Ψ the influence of these parameters on the stability of the closed loop system can be investigated

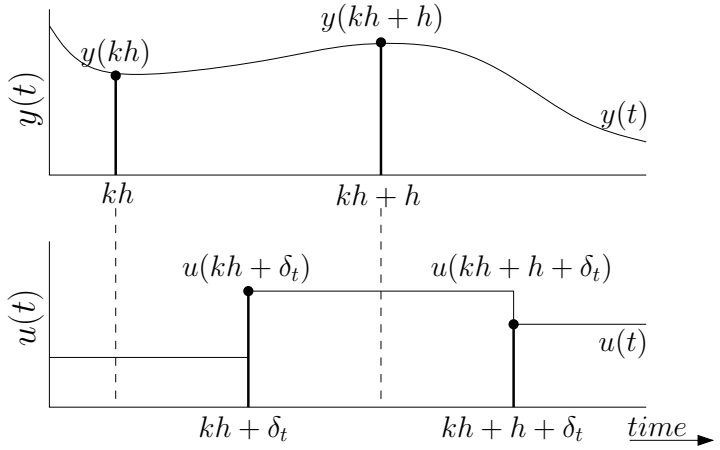


Figure 3.2: Timing of the input and the output of **H**.

If the total network induced delay $\delta_t > h$, the derivation of the model is straightforward. Suppose for example that the following condition for the total delay holds: $h < \delta_t < 2h$. Then the piecewise continuous control input $u(\tau)$ has again two different values over the sampling interval $[kh \quad kh + h]$.

$$\begin{cases} u(\tau) = -Ky(kh - 2h) & \text{for } \; kh < \tau < kh + \delta_t - h \\ u(\tau) = -Ky(kh - h) & \text{for } \; kh + \delta_t - h < \tau < kh + h. \end{cases} \tag{3.8}$$

The rest of the derivation can be done in the same manner. Note that in this case the dimension of the closed loop system matrix Ψ increases which makes the analysis of systems with larger delays more complex and the computational effort to analyze such systems analytically increases rapidly.

3.2 Stability analysis of a 1-dimensional system

With the model derived in the previous section, the influence of the delay δ_t, the sample-time h and the feedback gain K on the stability of the NCS can be analyzed for a given continuous-

time system **H**. As a first step, the following one-dimensional continuous-time system will be used for the sake of simplicity:

$$\begin{aligned} \dot{x}(t) &= x(t) + u(t) \\ y(t) &= x(t). \end{aligned} \tag{3.9}$$

Note that in open-loop this is an unstable system. If this continuous-time system is controlled by an output feedback $u(t) = -Kx(t)$ with no delay, it can be easily seen that the closed loop continuous-time system is stable if $K > 1$.
Fig. 3.3(a) shows the response of the NCS of Fig. 3.1 with continuous-time plant dynamics given by (3.9). The system has an initial state of $x(0) = 10$ and the state feedback controller aims to stabilize the system at the origin. The figure shows three different responses for three different values of the total network induced delay δ_t. Note that in these plots the continuous-time output $y(t)$ of **H** is given (see Fig. 3.1) and that a sample-time of $h = 0.1$ s is used.
As the time-delay increases, the settling time increases and therefore the performance of the system degrades. Time delays in the control loop are well known to degrade the performance of a control system [24], therefore this result is not surprising.
The other three sub-figures in Fig. 3.3 show a different phenomenon. In these figures, the gain $K = 30$. For $\delta_t = 0$ s, the system is unstable (Fig. 3.3(b)) as is the case for $\delta_t = 0.05$ s (Fig. 3.3(d)). For a delay of $\delta_t = 0.025$ s (Fig. 3.3(c)) however, the system is stable.
Although Fig. 3.3(a) shows the expected degradation of the performance of the controlled system due to the delay in the control loop, there is a situation where a certain amount of delay results in a stable system, whereas the same closed-loop system without delay is unstable.

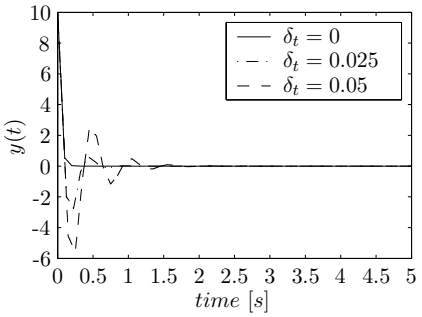
3.2.1 Eigenvalue analysis

In order to understand the positive influence of a certain amount of delay on the stability of the state-feedback controlled NCS as observed in the previous subsection, an analysis of the eigenvalues of the closed loop system matrix Ψ in (3.7) can give more insight in the encountered phenomenon. The matrix Ψ, subject to the system dynamics given in (3.9) is given by:

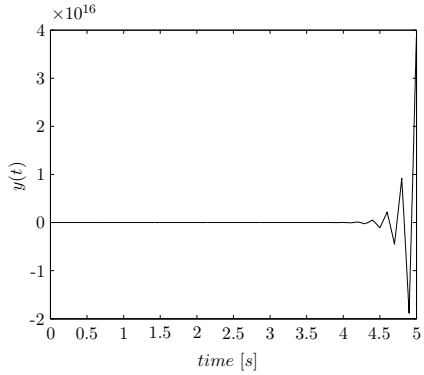
$$\Psi = \begin{bmatrix} e^h - \left(e^{h-\delta_t} - 1\right) K & \left(-e^h + e^{h-\delta_t}\right) K \\ 1 & 0 \end{bmatrix}. \tag{3.10}$$

In order for the system in (3.7) to be stable, the absolute value of the eigenvalues λ_i of Ψ have to lie inside the complex unit-circle, $|\lambda_i| < 1$. The eigenvalues can be obtained, using the characteristic equation $\det(\lambda I - \Psi) = 0$, see [8].

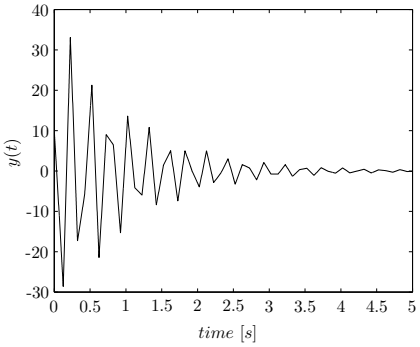
Fig. 3.4 to Fig. 3.7 show the eigenvalues for different total delay values δ_t and varying controller gain K. In the left graph of these figures, the absolute value of both eigenvalues is given. In the right graph, the eigenvalues are plotted in the complex plane. The start point ($K = 0$) is marked by an asterisk and the endpoint ($K = 30$) is marked by a triangle. Note that in Fig. 3.4 one eigenvalue is equal to zero for all K. Also note that both eigenvalues are zero around $K = 10$. This is the dead-beat solution
As can be seen in the figures, the interval of K for which the system is stable, is the largest



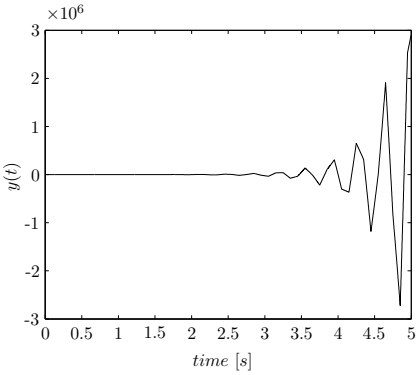
(a) Responses for $K = 10$.



(b) Response for $K = 30$ and $\delta_t = 0$ s.



(c) Response for $K = 30$ and $\delta_t = 0.025$ s.



(d) Response for $K = 30$ and $\delta_t = 0.05$ s.

Figure 3.3: Response of the NCS with the system in (3.9) with an initial state of $x(0) = 10$. $h = 0.1$ s.

for $\delta_t = 0.025$ s in Fig. 3.5. The minimum value of $|\lambda_i|$ however, increases with increasing delay δ_t.
For increasing delay, the eigenvalues exhibit an imaginary part. For a delay of $0.025s$ this results in a larger range of K for which the system is stable.
For $\delta_t = 0.05$ s and $\delta_t = 0.075$ s, which is 50 and 75 percent of the sample-time h, the imaginary part becomes so large that the eigenvalues $|\lambda_i|$ leave the unit circle faster and therefore, as δ_t increases, the range of K for which the system is stable decreases.

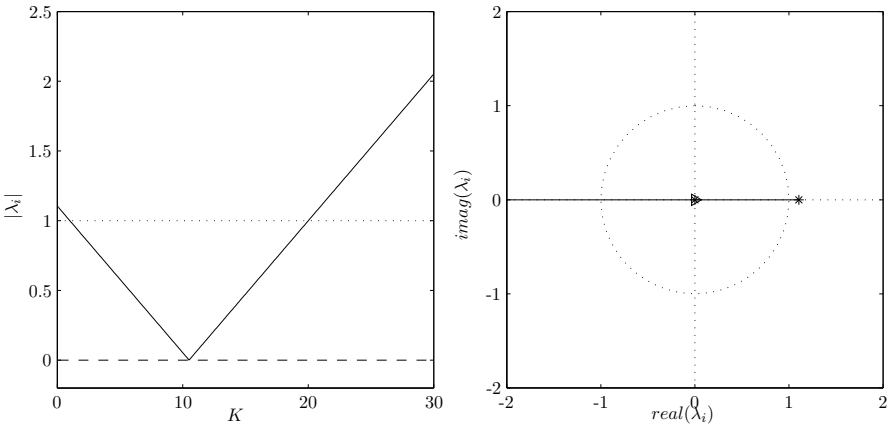


Figure 3.4: Two eigenvalues of (3.10) for $h = 0.1$ s and $\delta_t = 0$ s. λ_1: solid, λ_2: dashed.

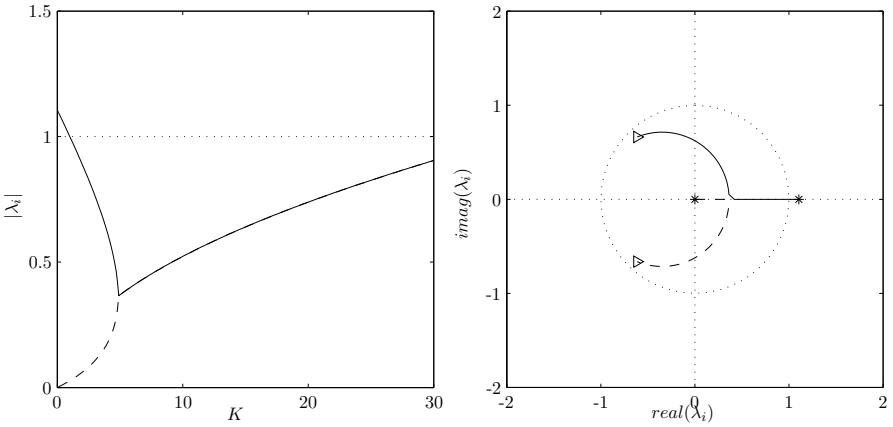


Figure 3.5: Two eigenvalues of (3.10) for $h = 0.1$ s and $\delta_t = 0.025$ s (25% of h). λ_1: solid, λ_2: dashed.

In order to gain a more physical understanding of this phenomenon, a frequency-domain analysis will be performed.

3.2.2 Frequency domain analysis

Recall from section 3.1 that the discrete difference equations that define the input/output behavior of the remote system (indicated by the dashed lines in Fig. 3.1) are given by:

$$\begin{aligned} x(k) &= \Phi x(k-1) + \Gamma_0 u(k-1) + \Gamma_1 u(k-2) \\ y(k) &= Cx(k). \end{aligned} \tag{3.11}$$

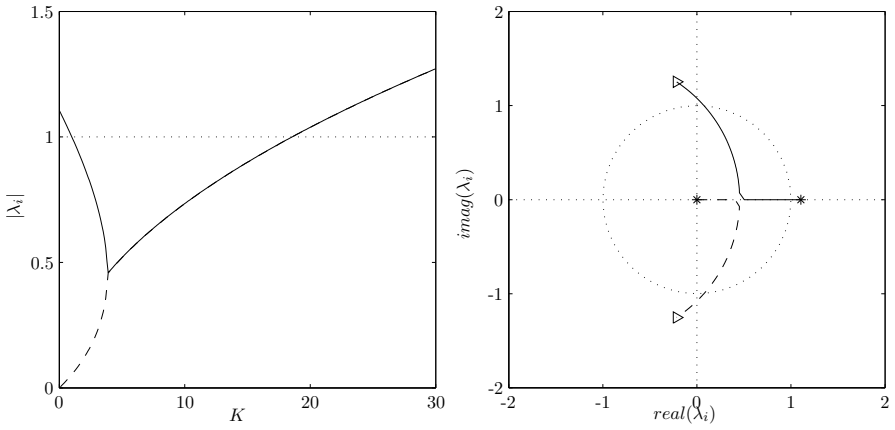


Figure 3.6: Two eigenvalues of (3.10) for $h = 0.1$ s and $\delta_t = 0.05$ s (50% of h). λ_1: solid, λ_2: dashed.

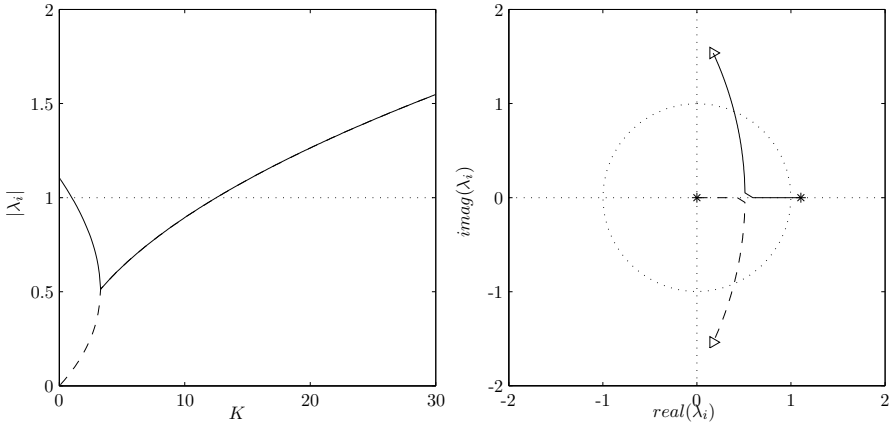


Figure 3.7: Two eigenvalues of (3.10) $h = 0.1$ s and $\delta_t = 0.075$ s (75% of h). λ_1: solid, λ_2: dashed.

Here $\Phi = e^{Ah}$, $\Gamma_0 = \int_0^{h-\delta_t} e^{A\eta} d\eta B$ and $\Gamma_1 = \int_{h-\delta_t}^{h} e^{A\eta} d\eta B$. Using the $\mathcal{Z}$-transform, this difference equation can be written in the $\mathcal{Z}$-domain as follows:

$$\begin{aligned} X(\mathcal{Z}) &= \Phi \mathcal{Z}^{-1} X(\mathcal{Z}) + \Gamma_0 \mathcal{Z}^{-1} U(\mathcal{Z}) + \Gamma_1 \mathcal{Z}^{-2} U(\mathcal{Z}) \\ Y(\mathcal{Z}) &= C X(\mathcal{Z}). \end{aligned} \tag{3.12}$$

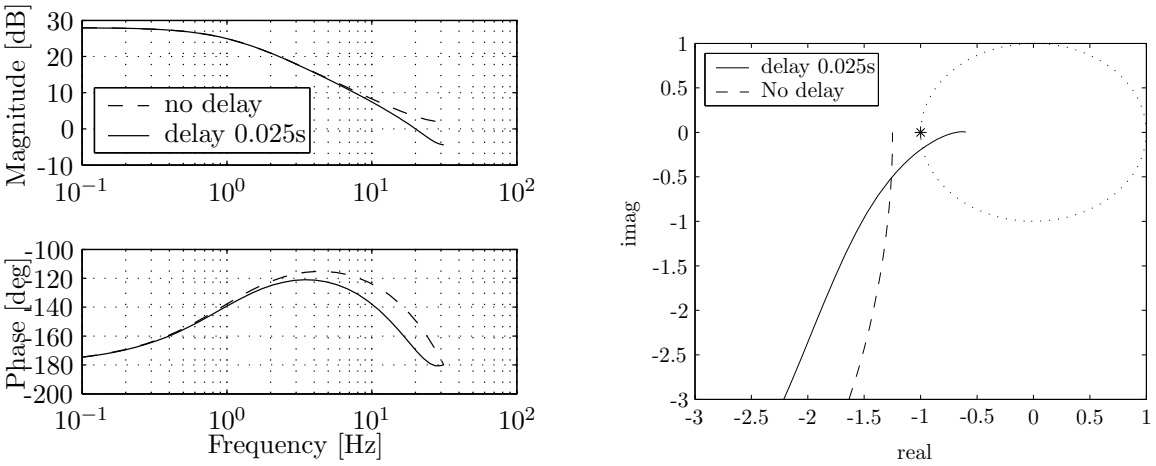
From (3.12), the discrete-time transferfunction of the sampled plant can be calculated:

$$\frac{Y(\mathcal{Z})}{U(\mathcal{Z})} = H_k(\mathcal{Z}) = C(\mathcal{Z}^2 I - \mathcal{Z}\Phi)^{-1}(\mathcal{Z}\Gamma_0 + \Gamma_1). \tag{3.13}$$

Again the one-dimensional system given in (3.9) is considered. The resulting discrete transferfunction of the sampled plant is given by:

$$H_\delta^*(\mathcal{Z}) = \frac{(e^{h-\delta_t} - 1)\mathcal{Z} + e^h - e^{h-\delta_t}}{\mathcal{Z}^2 - e^h \mathcal{Z}}. \tag{3.14}$$

Note that when the delay $\delta_t = 0$, the transferfunction reduces in order and the resulting transfer-function (3.15) is the same as would be obtained by discretizing the one dimensional



(a) Bode plot

(b) Nyquist plot

Figure 3.8: Bode and Nyquist plots for $K = 25$ and $h = 0.1$ s.

continuous-time system (3.9) with a zero-order hold discretization method:

$$H_\delta^*(\mathcal{Z}) = \frac{e^{h-\delta_t} - 1}{\mathcal{Z} - e^h}. \tag{3.15}$$

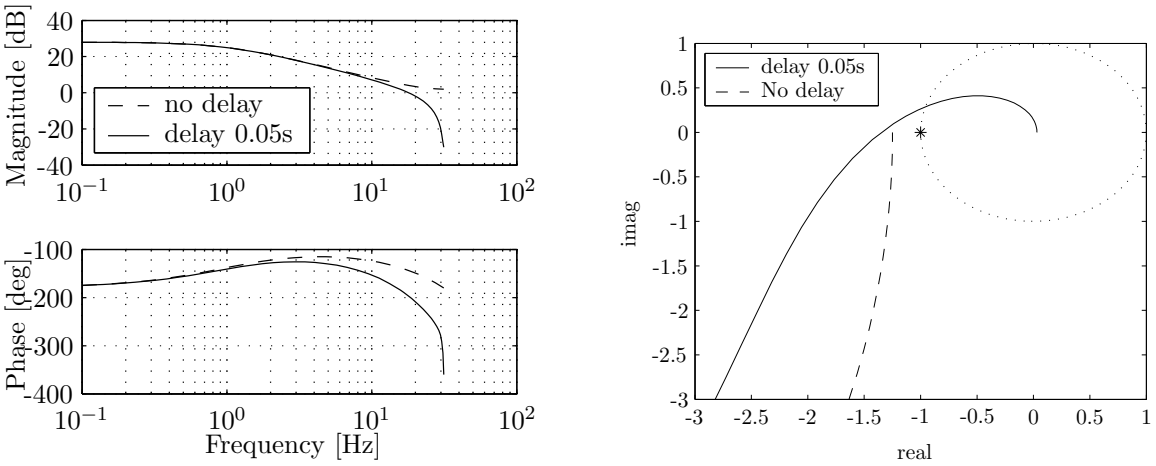
Again the controller **C** consists of an output feedback with gain K and the open-loop transferfunction is given by:

$$KH_\delta^* = K\frac{(e^{h-\delta_t} - 1)\mathcal{Z} + e^h - e^{h-\delta_t}}{\mathcal{Z}^2 - e^h\mathcal{Z}}. \tag{3.16}$$

Fig. 3.8 and Fig. 3.9 show the Bode plots and Nyquist plots of the open-loop system (3.16) with a gain $K = 25$ and a sample-time $h = 0.1$. Three different delay values are considered. In both figures the transferfunction for $\delta_t = 0$ is plotted. Fig. 3.8 also gives the transfer-function of the system with $\delta_t = 0.025$ (which is 25 % of the sample-time). In Fig. 3.9 the transfer-function of the system with $\delta_t = 0.05$ is drawn.
As observed in the previous section, the system with $\delta_t = 0$ is unstable. This also can be seen in the Bode and Nyquist plots in Fig. 3.8. In the same figure, the open-loop transfer function for $\delta_t = 0.025$ is drawn. As can be seen in the figure the system is stable. Although the delay introduces a phase lag, the gain is lower compared to the no-delay case. Therefore, there is still some phase margin at the zero dB crossing of the gain.
For the case where $\delta_t = 0.05$ depicted in Fig. 3.9, the gain is also lower than in the no-delay case. The phase lag due to the delay however is so large that the system is unstable.

3.2.3 An analytic stability bound

Due to a slight amount of delay in the control loop, the open-loop gain of the discrete time NCS as formulated in (3.7) appears to be lower than that of the system without delay. Although the delay introduces a phase lag, the system is stable for a limited amount of delay.
In order to determine the stability region for the considered first order system, an analytic stability bound can be obtained. With the so-called Jury test, the discrete equivalent of the



(a) Bode plot

(b) Nyquist plot

Figure 3.9: Bode and Nyquist plots for $K = 25$ and $h = 0.1$ s.

Routh/Hurwitz criterium, an analytic expression for the stability bound of the NCS can be found[1].
The Jury test provides two stability conditions:

$$\begin{cases} 1 - \left(Ke^{h} - Ke^{h-\delta}\right)^2 & > 0 \\ 1 - \left(Ke^{h} - Ke^{h-\delta}\right)^2 - \frac{\left(Ke^{h-\delta} - K - e^{h} - \left(Ke^{h} - Ke^{h-\delta}\right)\left(Ke^{h-\delta} - K - e^{h}\right)\right)^2}{1 - \left(Ke^{h} - Ke^{h-\delta}\right)^2} & > 0. \end{cases} \tag{3.17}$$

From (3.17) a stability bound for δ_t can be computed as a function of the controller gain K and the sample time h. The matrix Ψ in (3.10) has all eigenvalues inside the unit circle if the following relation holds:

$$h + \log\left[\frac{2K}{Ke^{h} + e^{h} + K + 1}\right] < \delta_t < h + \log\left[\frac{K}{Ke^{h} - 1}\right]. \tag{3.18}$$

[1] More information of the Jury test and an implementation in a MATLAB m-file can be found in appendix C.

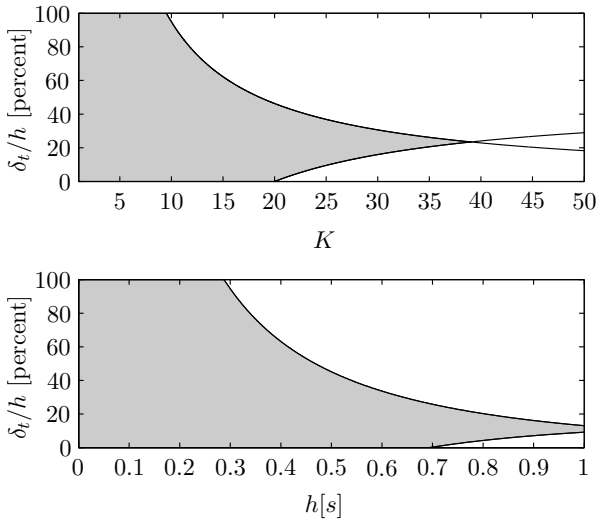


Figure 3.10: Stability region for varying h and K and δ_t. Upper figure $h = 0.1$ s, lower figure $K = 5$. The shaded area indicates the stable region.

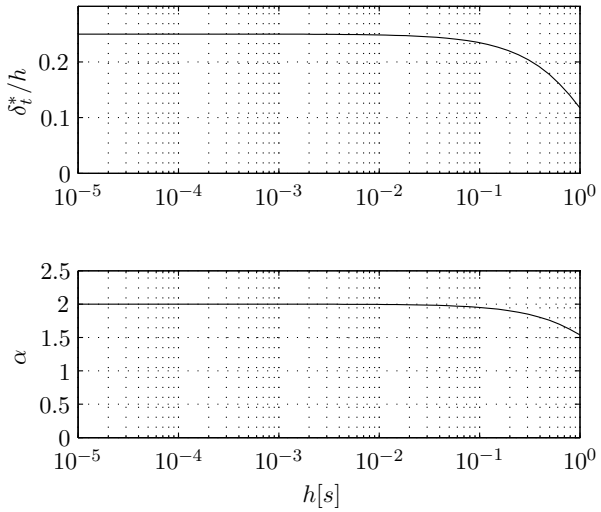


Figure 3.11: δ_t^* and α as a function of the sample time h.

Fig. 3.10 shows the stability of the system for varying gain, sample-time and network delay. The area for which the system is stable is shaded. As can be seen in the figure, there is a value for δ_t for which the set of K values which give a stable system is maximal (the intersection point of the two stability boundaries in the upper figure of Fig. 3.10). This value, which will be called δ_t^* can be computed by equating both boundaries given in (3.18):

$$\delta_t^* = h + \log\left(\frac{3 + e^h}{2e^h + 1 + e^{2h}}\right). \tag{3.19}$$

The upper graph of Fig. 3.11 shows δ_t^*/h as a function of h. The figure indicates that for decreasing sample time h, δ_t^*/h converges to a constant value. For $h \to 0$ the delay value δ_t^*/h

can be computed using (3.19):

$$\lim_{h\to 0}\left[\frac{\delta_t^*}{h}\right] = \frac{1}{4}. \tag{3.20}$$

Appendix B gives the derivation of this limit.

Let K_{max} be the maximum gain for which the system is still stable. Obviously, K_{max} is a function of δ_t and h. For K_{max} evaluated for $\delta_t = 0$ and $\delta_t = \delta_t^*$ the following relation holds:

$$K_{max}|_{\delta_t=\delta_t^*} = \frac{3e^{-h}+1}{1+e^{-h}}\, K_{max}|_{\delta_t=0} = \alpha\, K_{max}|_{\delta_t=0}\,. \tag{3.21}$$

The lower graph of Fig. 3.11 shows the parameter α as a function of h. For $h \to 0$ the value of α converges to a constant value:

$$\lim_{h\to 0}[\alpha] = 2 \tag{3.22}$$

When the system has a delay δ_t of 25 percent of the sample-time h, the gain K can be doubled to reach the stability boundary compared to the same system with no delay in the considered case.

3.3 Stability of a two dimensional model

In the previous section, the stability of an NCS has been investigated using the model derived in section 3.1. For the sake of simplicity, a simple one-dimensional model has been used. The use of a one-dimensional model allows us to use several techniques to analyze the stability of the closed-loop system, because the limited dimension of the state of the NCS model confines the computational cost required to apply these techniques. The physical relevance of the used system however is limited.
Therefore, a stability analysis will be performed using a system that can directly be related to a motion system.

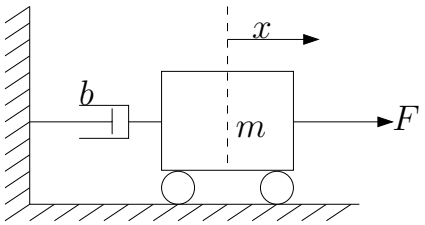


Figure 3.12: An actuated moving mass with viscous damping.

Consider the cart given in Fig. 3.12. The cart with mass m experiences a linear viscous damping force $b\dot{x}$ in opposite direction of the applied force F, where b is the viscous damping coefficient. This type of system, a moving mass which can be positioned by applying a force

F, is a common example of a motion control system.
Suppose that both the position x as well as the velocity $\dot{x}$ of the cart can be measured. The dynamics of the moving cart in state space notation is then given by:

$$\begin{aligned} \begin{bmatrix} \dot{x} \\ \ddot{x} \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ 0 & -\frac{b}{m} \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{m} \end{bmatrix} u \\ y &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \end{bmatrix}, \end{aligned} \tag{3.23}$$

where the input u is the force F applied to the mass in x-direction.
Suppose $b = m = 1$, and this system is controlled over a network by a proportional output feedback controller $u = -Ky$, where K is given by $[K_1 \;\; K_2]$. Note that in the continuous-time case with no delay, the closed loop system can only be stabilized if $K_1 > 0$.
When $K_2 = 0$, the closed-loop system matrix Ψ from (3.7) of the NCS with the dynamics of the plant given by (3.23), is given by:

$$\Psi = \begin{bmatrix} 1 - K_1 \left(e^{-h+\delta_t} + h - \delta_t - 1\right) & -e^{-h} + 1 & K_1 \left(-e^{-h} + e^{-h+\delta_t} - \delta_t\right) & 0 \\ -K_1 \left(-e^{-h+\delta_t} + 1\right) & e^{-h} & K_1 \left(e^{-h} - e^{-h+\delta_t}\right) & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}. \tag{3.24}$$

Due to the fact that the dimension of the discrete-time system equations increases with an order two, the computation of the stability borders using the Jury test becomes too complex. Therefore, the stability region is obtained by calculating the eigenvalues for a grid of points, characterizing combinations of K_1 and δ_t. The stable region for this case is drawn in Fig. 3.13(a), the used sample-time is $h = 0.1$ s.
As can be seen in Fig. 3.13(a), the system can be stabilized for a gain $K_1 > 0$ up to $K_1 = 20$ for the situation where $\delta_t = 0$. As the network induced delay increases, the region for K_1 for which a stable system results, narrows. Fig.3.14 shows the root loci for this system for several values of the delay ($\delta_t = \{0, 0.005 \ldots 0.1\}$). The root loci are computed for varying gain K_1 ($K_1 = \{0, 0.1 \ldots 25\}$) and all for a sample-time $h = 0.1$ s. An asterisk marks the start of the root locus where $K_1 = 0$, the end of each root locus ($K_1 = 25$) is marked with a triangle . Note that the x-axis and y-axis as well as the unit circle are indicated by the dotted lines.
The left figure shows the loci of the first two roots λ_1 and λ_2. These roots are real and lie inside the unit circle for all considered combinations of K_1 and δ_t.
The right figure shows the loci of the other two roots. Both roots are real for $K_1 = 0$, $\lambda_3 = 0$ and $\lambda_4 = 1$. For $K_1 > 0$ both roots lie inside the unit circle and for $K_1 > 1$, λ_3 and λ_4 are complex conjugates. The two arrows give the direction of increasing delay δ_t. The stable region given in Fig. 3.13(a) is completely determined by λ_3 and λ_4.

Now suppose $K_1 = 1$ and the system is controlled by a state-feedback controller $u = -x - K_2\dot{x}$. The closed loop NCS matrix Ψ is then given by:

$$\begin{bmatrix} 2 - h + \delta_t - e^{-h+\delta_t} & -e^{-h} + 1 - (h - \delta_t + e^{-h+\delta_t} - 1)K_2 & -e^{-h} - \delta_t + e^{-h+\delta_t} & (-e^{-h} - \delta_t + e^{-h+\delta_t})K_2 \\ e^{-h+\delta_t} - 1 & e^{-h} - (-e^{-h+\delta_t} + 1)K_2 & e^{-h} - e^{-h+\delta_t} & (e^{-h} - e^{-h+\delta_t})K_2 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.25}$$

The stability region for this case is given in Fig. 3.13(b) also for a sample-time of $h = 0.1$s. As can be seen, the same phenomenon in the stability region can be observed as was the case with the one-dimensional example. There is a region where the system is only stable for a certain amount of delay δ_t. Fig. 3.15 shows the root loci for this case subject to $K_2 = \{0, 0.1 \ldots 45\}$ and $\delta_t = \{0, 0.005 \ldots 0.1\}$. Again the arrows indicate the directions of increasing delay δ_t.

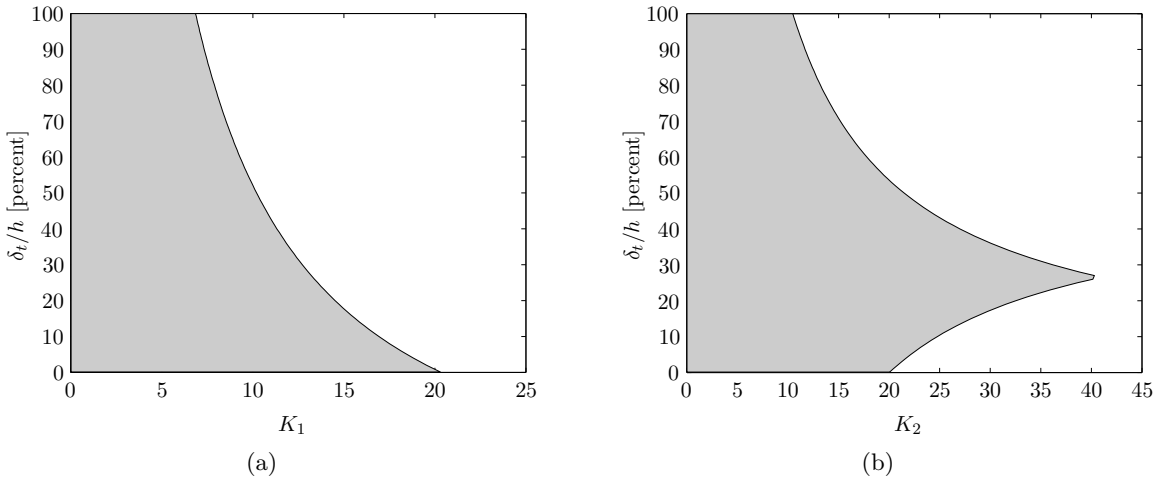


Figure 3.13: Stability of the NCS subject to the 2-D system. The shaded area indicates the stable region for $h = 0.1$ s.

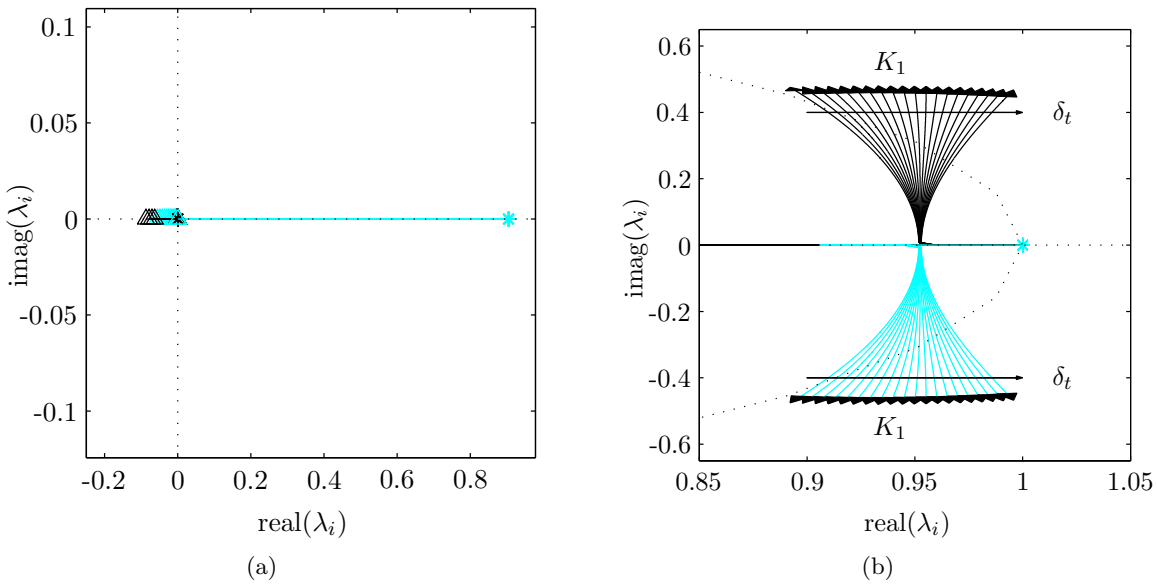


Figure 3.14: The root loci for the position $u(k) = -K_1 x(k)$ feedback for varying K_1 and increasing delay value δ_t. $K_2 = 0$ and $h = 0.1$ s.

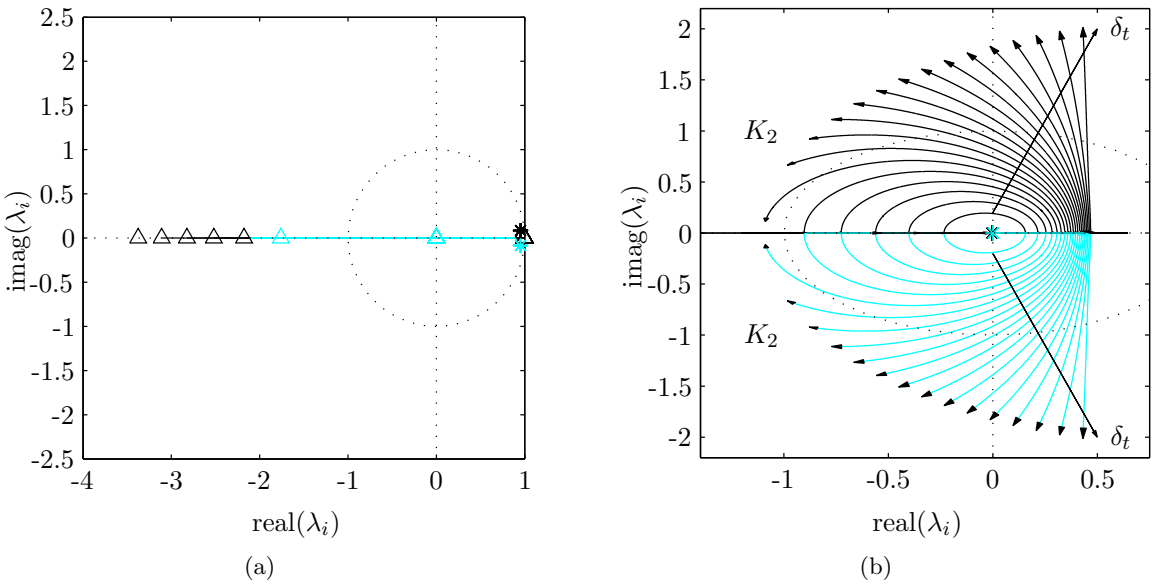


Figure 3.15: The root loci for the full state feedback $u(k) = -K_1 x - K_2 \dot{x}$ for varying K_2 and increasing delay value δ_t. $K_1 = 1$ and $h = 0.1$ s

3.4 Discussion

In this chapter, a discrete-time model of an NCS with output feedback has been derived under the assumption that the network induced delays are constant and the total network induced delay δ_t is smaller than the sampling interval h.
By means of this model, the stability of an NCS is investigated. As a first step, a one-dimensional model is used to analyze the stability of the closed-loop system through eigenvalue analysis and frequency domain techniques. Using the Jury test, also an analytic stability bound is obtained.
To add some more physical meaning to the analysis, a two-dimensional system is introduced and, although the NCS model considerably increases in complexity for this two-dimensional case, an eigenvalue analysis can be used to analyze the stability of this system.
Delay in a control loop is well known to degrade the system performance which can also be seen when considering the poles of the closed-loop system. An interesting observation however is, that a certain amount of delay in the control-loop can have a positive effect on the stability of the closed-loop system. In the one-dimensional case this phenomenon appeared with a feedback $u = -Kx$. In the two dimensional system this phenomenon appeared when using a state feedback $u = -K\dot{x}$.
In general, an n-dimensional system controlled using a feedback $u = -Kx^{(n-1)}$ translates in the discrete-time formulation with a delay $d_t < h$ into the encountered phenomenon. Note that $x^{(n-1)}$ is the $(n-1)$th time derivative of x.
To test wether or not this phenomenon can be reproduced in practice, an experimental setup has been created to validate the obtained results. In the next chapter, the created setup will be discussed. In chapter 5 the conducted experiments will be covered.

Chapter 4

An experimental NCS setup

In this chapter, the experimental setup used in this research will be discussed. The setup is a mobile robot that can be purchased as an easy-to-use robotics platform for education and research. The standard configuration of this robot has some major drawbacks which make it not directly applicable for the research in this report or control related research in general. Some customization of the standard configuration, however, can solve these drawbacks. In this chapter, the work done to fit the mobile robot to our needs is discussed. In section 4.1, the mobile robot will be introduced. The features of the standard configuration will be discussed briefly as well as the reason why this setup can be suitable for NCS experiments. Furthermore, the main drawbacks of the standard configuration and the necessary adaptations to overcome these drawbacks are discussed.
In section 4.2, the operation of the microcontroller on-board the robot is discussed to provide some insight in the possibilities offered by the used microcontroller. In section 4.3 the implementation of the required adaptations is discussed. Two different configurations are developed, one configuration for control experiments (i.e. trajectory tracking) in general and one configuration with some specific options, tailored to this research.
In section 4.4, the implementation of the main controller on the desktop PC is discussed as well as some time-keeping issues involved in this implementation. In the last section of this chapter, some specific issues that are of concern for the experiments covered in the next chapter are discussed.

4.1 The Trilobot mobile robot

The experimental setup used in this research is the Trilobot mobile Robot from Arrick Robotics shown in Fig. 4.1 [2]. Trilobot is a mobile robot that consists of a base with two wheels driven by DC motors (drive-wheels) and a castor wheel. There is an optical encoder attached to each driven wheel. The Trilobot is normally powered by a battery pack but in the setup used for this research, the battery pack is replaced by a rechargeable one. A controllable gripper is mounted between the two drive-wheels. Furthermore, the base is equipped with eight independently readable whiskers and a water sensor.
The top of Trilobot is called the mast and there are several sensors and controllable features mounted on the mast, including:

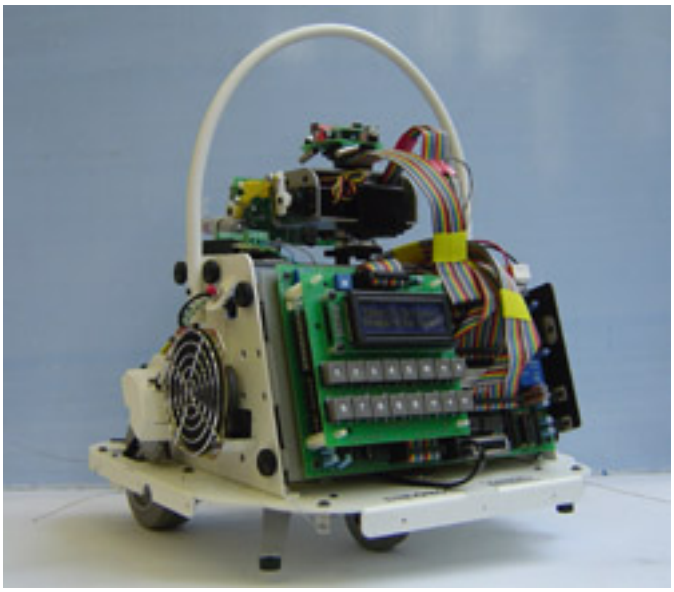


Figure 4.1: The Trilobot mobile robot.

- Electronic Compass;
- 4 light level sensors;
- Temperature sensor;
- Tilt sensors;
- IR communication transmitter;
- Green and red LED.

In front of the mast, a controllable head is mounted, driven by three servos. The features mounted on this head are:

- IR communication receiver;
- Sonar range finder;
- Passive infrared motion detector;
- Headlight;
- Sound detection;
- Laser pointer.

All the features of Trilobot are controlled via a controller board with a main microcontroller that runs Trilobot's operating software. Two microcontrollers are used as intelligent co-processors and are connected to the main processor using serial connections. There is a

display and a keypad available to interface the software.
The software that runs on Trilobot's main controller has several operating modes. The first mode, the joystick control mode, can be used to control the drive-wheels, the head and the gripper functions using a joystick that can be connected to the joystick port on the main control board.
The second mode, the IR control mode, can be used to control the robot with the Infra-Red remote control that comes with Trilobot. Many of Trilobot's functions can be controlled using the remote control when Trilobot is in IR control mode.
The third mode, the console command mode, can be used to control Trilobot from a desktop PC using a serial connection. Making use of a high level programming language, simple ASCII commands can be sent to Trilobot from the desktop PC using the serial communication (RS-232) protocol. The serial connection between Trilobot and the desktop PC can be established using a cable or a wireless serial link. The response of Trilobot to various commands is sent back to the desktop PC using the same serial connection. There is a wide range of commands available that can be used to control the drive-wheels, gripper and head and to request data of all the sensors on board Trilobot.
Another very powerful function of Trilobot is to run programs written by the user that can be uploaded to Trilobot's microprocessor and run directly from the available memory of Trilobot's processor. Using a user program, Trilobot can operate autonomously without any communication with an external PC. Furthermore, a custom control mode can be developed to communicate with an external PC. User programs can be written in C or Assembly language and have to be compiled and assembled to an Intel hex file. When programming in Assembly, it is easy to call several routines readily available in the system software [2].

Previous research pointed out that by using the user programs, the effectiveness concerning speed and accuracy in the operation of Trilobot can be improved to a large extent as opposed to making use of the console command mode [11]. Avoiding the use of the predefined functions, which cause a lot of overhead, can improve the performance even more. Suggestions for improvement are pointed out in the next section.
The user programs discussed in this report are written in Assembly and are listed in Appendix F.

4.1.1 NCS research using Trilobot

To validate the results obtained in the previous chapter experimentally, a setup has to be created that consists of a plant and a separate controller connected using some kind of data network that induces a delay. The sample-rate of the controller and the plant have to be manually adjustable.
It is possible to communicate with Trilobot using a desktop PC with serial connection to Trilobots main processor. Software that runs on the main processor can interface the actuators and sensors on-board Trilobot. The PC can be regarded as a controller and Trilobot as the remote system. The serial connection is a type of network connection where the data transfer rate can be set manually. In brief, Trilobot could be a suitable setup for NCS experiments.

Trilobot has many sensors and actuators, but the most appealing ones are the two drive-motors with encoders. Controlling the angle or the angular velocity of a DC motor is a common example in control engineering. So when the two drive-motors can be controlled separately, a useful setup results for control related research. With this setup, trajectory

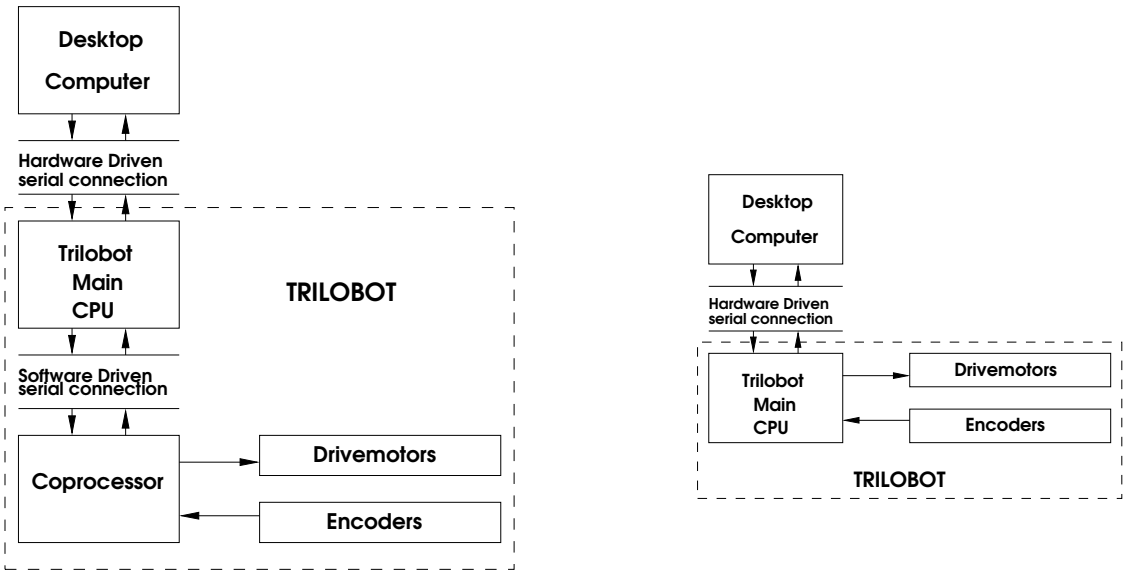
planning experiments and other mobile robotics experiments can be performed. The drive-motor controller can run on an external PC connected to Trilobot using the serial connection, which makes the setup suitable for NCS experiments. Furthermore, the drive-motor controller can run on the on-board embedded microcontroller which makes the setup an embedded control system, which is suitable for embedded motion control research and education.

4.1.2 Necessary adaptations

Although all the features that are necessary to qualify Trilobot, in combination with a desktop PC, as an NCS are available, it can be pointed out that there are some major drawbacks in the standard configuration of Trilobot. Trilobot's standard configuration is depicted in Fig. 4.2(a). The two drive-motors can only be driven by sending commands from the main processor to one of Trilobot's coprocessors. With these predefined commands Trilobot can drive forward for a specified distance with a fixed speed or turn a number of degrees with a specified radius, again with a fixed speed. The low-level controllers used to control the drive-motors run on the coprocessor. The coprocessor keeps track of the traveled distance of each drive-motor by counting the pulses generated by the encoder circuitry and based on this information the motors are driven to perform the requested action. The software (i.e. the low-level controllers) that runs on the coprocessor cannot be changed. The encoder data can be requested from the coprocessor by the main processor, but the returned data are the sum of the counts of both encoders.
To make the Trilobot setup suitable for NCS research or control related research in general, it should be possible for the user to close the control loop either on-board Trilobot or over the serial line. This means that the drive-motor controllers have to reside on Trilobot's microprocessor or on the desktop PC. In order to achieve such a configuration, the data of both encoders have to be imported into the main microcontroller separately. Furthermore, it must be possible to control the input voltage of both drive-motors separately from the main processor.
In order to achieve such a setup, the drive-motors and the encoders have to be disconnected from the coprocessor in order to connect them directly to the main processor, creating a configuration as given in Fig. 4.2(b). In order to understand how to carry out the necessary adaptations, a brief explanation of the operation and special features of the main microcontroller on-board Trilobot is given in the next section.

4.2 The 8052 Microprocessor

In order to be able to interface the drive-motors and encoders using the main processor, some knowledge on the operation and features of the microcontroller on-board Trilobot is required. First, the basic operation of the microcontroller will be discussed. After that some special on-board features are discussed. This section ends with a brief explanation of the specific configuration of the microcontroller on-board Trilobot and the ports available to the user.



(a) Standard drive-motor configuration.

(b) Desired drive-motor configuration.

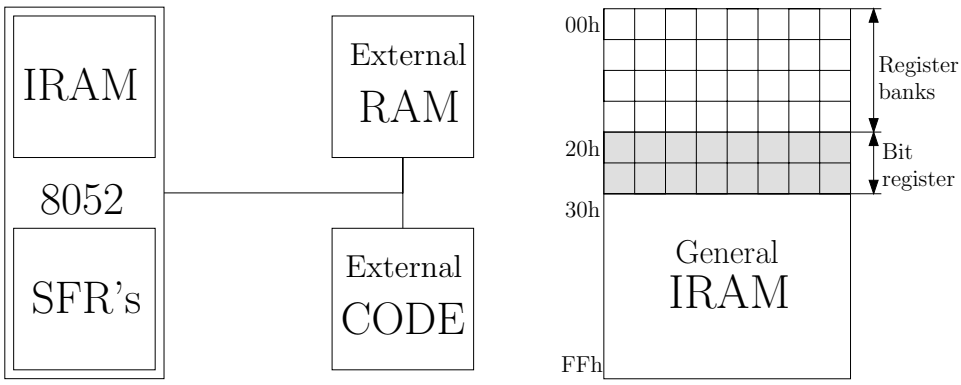
Figure 4.2: Necessary adaptation in the drive-motor configuration.

4.2.1 Basics of the 8052 microprocessor

The main processor of Trilobot is a Winbond W78E52B micro-controller unit (MCU) [30]. This MCU is based on the 8052 core licensed from Intel. The core refers to the instruction set and special function registers (SFRs), special data registers that can be used to control the on-chip features of the MCU. The 8052 core MCU is frequently used in commercial products and consequently many other manufacturers produce 8052 compatible MCUs, e.g. Philips, Atmel, Dallas Semiconductors and of course Intel [25, 31].

The 8052 core MCU (which will be referred to in the remainder of this thesis as 'the 8052') is an 8-bit MCU. Consequently it has an instruction set that defines 256 instructions. Instructions are operations that can be carried out by the processor, such as addressing memory and performing mathematical operations.
The 8052 operates based on an external crystal. This crystal is an electronic device that emits pulses at a fixed frequency when power is applied. The crystal frequency of the 8052 used on-board Trilobot is 11.059 MHz. This seems to be a strange frequency, but it is very convenient to set standard serial port Baud-rates, as will be discussed in section 4.3. The 8052 uses the crystal to synchronize its operations. Effectively, the 8052 uses so-called machine cycles. A single machine cycle is the minimum amount of time in which a single 8052 instruction can be executed. Many instructions however take multiple cycles. In the 8052, a machine cycle takes 12 crystal pulses. The memory of the 8052 can be accessed by 8-bit addresses which can be quickly stored and manipulated by the 8-bit processor.

The 8052 has 4 8-bit I/O ports, numbered P0 to P3, which means that there are 32 I/O pins. Every port has a dedicated SFR. Changing the status of the output pins comes down to writing an 8-bit value to the dedicated SFR address. All the bits of a port SFR are also bit addressable, so it is possible to set or clear only one bit of a port. Some of the output bits



(a) Memory configuration of the 8052.

(b) Internal RAM of the 8052.

Figure 4.3: Internal and external memory configuration of the 8052.

can have a special function. They can for example be used to address external memory or they serve as I/O lines of the serial port. If no external memory or serial port is connected, these ports can be used as general purpose I/O ports.

The 8052 can address three types of memory, i.e. internal RAM (Random Access Memory) which resides on-chip, referred to as IRAM, external RAM and external code memory. The IRAM consists of 256 bytes and another 128 bytes of the so-called special function registers (SFRs) which can be used to configure all features of the 8052. To effectively program the 8052 it is necessary to have a basic understanding of these memory types. Fig. 4.3(a) gives a schematic representation of the memory configuration of the 8052.

Internal RAM The 8052 has 256 bytes of IRAM that can be found on-chip; so it is the fastest RAM available and it is very flexible in terms of reading, writing and modifying its content. Internal RAM is volatile, so it will be cleared when the MCU is reset. A schematic representation of internal RAM is given in Fig. 4.3(b). The first 32 bytes (00h-1Fh in hexadecimal representation) are reserved for four register banks each of 8 bytes. The next 16 bytes (20h-2F) are reserved as bit memory containing 128 bits. The 208 bytes of internal RAM remaining (30h-FFh) can be used to store data that have to be accessed frequently or at high speed. This area is also utilized by the processor to store the operating stack.

128 bytes of the on-chip memory of the 8052 microcontroller are used as SFRs (see Fig. 4.3(a)). These SFRs are registers in memory that control specific functionality of the 8052 processor. It is important that during program execution this area of RAM is not used to store data because the settings of the MCU will be changed during operation, which leads to unexpected results.

The 8052 uses eight 8-bit R-registers that are used in many of the instructions that can be performed, e.g. manipulating values and moving data. These R-registers are numbered from 0 up to 7 and together they form a register bank. There are four of these register banks. On startup of the processor, register bank 0 (00h-07h) is used by default. One can instruct the processor to use another register bank. The register banks that are not in use can be used as extra internal RAM. It is important to know which register bank the processor uses. If variables are stored in the addresses of the

register bank in use, those variables will be overwritten while the processor carries out instructions.

Bit memory provides 128 bits that can be accessed separately using special instructions. These 128 bits can also be used as another 16 bytes of memory.

External code memory The 8052 allows up to 64K of external code memory. On these 64K of external code memory only read operations can be performed. Usually the external code memory is provided in the form of EPROM (electrically programmable read only memory) which implies that this memory is read only. To change the code in the EPROM, a special device called a ROM burner is required. Code memory is mostly used to hold the actual program that has to be run.

External RAM The 8052 also supports external RAM. External RAM has a read/write functionality. Accessing external RAM costs more machine cycles than accessing internal RAM. This makes external RAM slower. The capacity however is much larger than the capacity of internal RAM. While internal RAM is limited to 256 bytes, the 8052 supports external RAM up to 64K.

4.2.2 Timers

The 8052 is equipped with three timers. These three timers can be configured and read separately using the SFRs dedicated to a timer. All three timers can perform three different types of actions:

Keeping time The most obvious function of a timer is keeping time. When configured for this purpose, the timer increments every machine cycle. With an 11.059 MHz crystal and machine cycles of 12 crystal pulses the timer increments with a frequency of 0.9216 MHz.
The timer can be configured as 8-bit timer, 13-bit timer or 16-bit timer. In 8-bit timer mode, the timer counts up from 0 to 255 and then it overflows back to zero. A special flag is set on overflow and even an interrupt can be triggered on overflow. Furthermore an auto reload value can be set. This means that when the timer overflows it will not be set back to zero, but to a value specified in the program. This functionality of a timer can be used to effectively obtain a fixed sample-rate.

Count events Another way a timer can be configured is to count events on an external processor pin. Processor pin P3.4 (bit 4 of port 3) and P3.5 can be monitored by timer 0 and timer 1, respectively. When configured as event counter, the timer will not increment every machine cycle, but it will increment if it detects a high to low transition on the corresponding pin.

Serial port Baud-rate generation The 8052 also has an integrated serial port. To set the Baud-rate (i.e. the communication speed) of this serial port, also a timer can be used. When no Baud-rate is set, the serial port uses the crystal frequency to clock out each bit with a fixed rate. This rate can also be set using one of the timers. As mentioned before, a crystal rate of 11.059MHz is suitable to set standard Baud-rates often used in serial communication [25, 31].

4.2.3 Interrupts

An interrupt is a special event that interrupts the normal program flow. There are three types of events that can trigger an interrupt:

1. Timer overflow;
2. Reception/transmission of data over the serial line;
3. External event on a specific processor pin.

When an interrupt occurs, the normal program flow is put on hold and the program jumps to a specific address in memory. At this specific address, a subroutine can be placed that executes a function related to the occurred event. After this so called interrupt service routine (ISR) is finished, the program resumes where is was interrupted.
The ability to interrupt normal program execution when certain events occur makes it much easier and much more efficient to handle certain conditions. Suppose for example that in every program loop the serial port has to be checked in order to detect incoming data. A lot of machine cycles would be waisted to check for an event that does not happen that often.

4.2.4 Serial communication

One of the most powerful features of the 8052 is the integrated UART (Universal Asynchronous Receiver/Transmitter), or serial port. The integrated serial port makes it very easy to read values from and write values to the serial port. If it were not for the integrated serial port, writing a byte to a serial line would be a tedious process requiring turning on and off one of the I/O lines in rapid succession to properly clock out each individual bit. With the integrated UART one does not have to worry about this low level transmission, which saves a lot of processor time and code.
Using the serial SFR, the operation mode and Baud-rate can be configured. Once configured data can be written to an SFR, which is used as a serial buffer. The UART converts the data bytes to a binary stream that is suitable to send over the serial connection. Incoming data will be transformed into bytes by the UART and loaded in the serial SFR. Therefore, reading an incoming value comes down to reading the same SFR. The 8052 will set a flag and generate an interrupt (if configured to do so) when it has finished sending a byte and when it receives a byte. In that manner, the received data can be processed as soon as they arrive at the serial port.

4.2.5 Configuration on-board Trilobot

Fig. 4.4 shows a schematic of the external memory of Trilobot. The upper 32K of code memory holds the system software. On startup this code is executed. The bottom part of the EPROM also hold the interrupt vectors, according to the standard addresses for the interrupt routines [30]. In the rest of code space some speech data that are used by the speech processor, another on-board expansion, reside. The upper part of RAM is used for some hardware I/O buffers. The lower 32K of RAM can be used by the user. The system software offers functionality to upload user programs to this free 32K of RAM. The user program can then be executed from

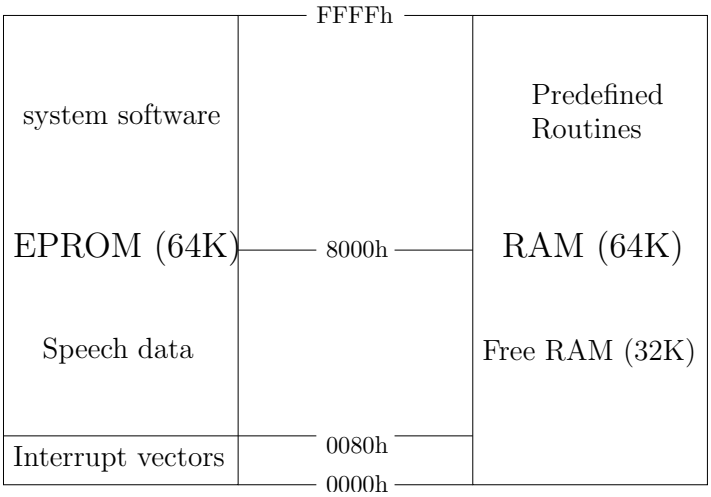


Figure 4.4: Memory map of Trilobot's controller.

a RAM address that can be specified after the upload.
It is important to note that the interrupt vectors then correspond to addresses in external RAM. The interrupt vectors are copied from code space to external RAM upon startup. This means that the interrupt routines of Trilobot's standard software are executed whenever an interrupt occurs. So it is important to change the code in the interrupt routines after the startup of the user program.
Most processor pins are in use to interface all the on-board devices. There are two processor pins, P3.4 and P3.5, which are not in use and they can be used for expansion.

4.3 Trilobot as remote system

As explained in section 4.1, the drive-motors have to be connected to the main controller in order to be able to control both motors separately. The same holds for the encoders. In this section, the new configuration will be discussed. First, the new circuitry for the drive-motors will be discussed together with the software required to drive the motors. Next, the changes in the encoder circuitry will be discussed and how the circuitry can be interfaced in the software. At the end of this section, the software required to send data to the desktop PC and to receive data from the desktop PC using the serial port will be discussed.

4.3.1 Drive-motor circuitry

The two drive-motors of Trilobot are permanent magnet DC motors (see table 4.1). The speed of the motors can be varied by changing the input voltage. One of the easiest ways to generate an analog output voltage from a digital value is by pulse-width modulation (PWM). In PWM a high frequency square wave is generated by a digital output by continuously switching the digital output bit on and off. This digital signal is often a CMOS or TTL level signal. In order to obtain a suitable input signal for the analog electronics to be driven (in this case a DC motor), the digital output is fed through some additional circuitry called an H-bridge, which is powered by an external power source. H-bridge circuits are readily available as IC's

Table 4.1: Characteristics of Trilobot's drive-motors.

Type	Permanent magnet DC motor
Supply Voltage	12V
Current	0.5A no load, 1.5A full load
Torque	0.79 Nm
Speed	73 rpm full load

and some of them are specially designed for motion control applications. The output voltage

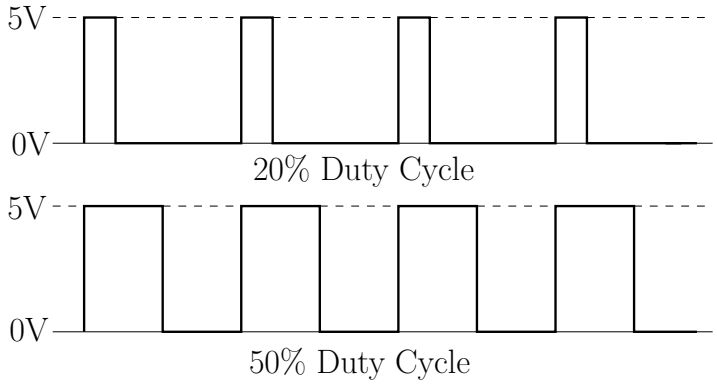


Figure 4.5: 20% and 50% duty cycle PWM signals.

of the H-bridge circuit can be altered by changing the duty cycle of the PWM signal. The duty cycle is the period the square wave stays high during one cycle. Fig. 4.5 shows two PWM signals, one with a 20% duty cycle and one with a 50% duty cycle. In case of the 50% duty cycle, the output of the H-bridge circuit is 50% of the input power. The H-bridge IC used for Trilobot's drive-motors is the LMD18200 by National Semiconductors. It is a device specially designed for DC motor and stepper motor control.

The used circuit is given in Fig. 4.6, where M_1 and M_2 are the two drive-motors. The PWM input is translated to a positive input voltage for the DC motor. A zero duty cycle generates a zero input voltage and a 100% duty cycle generates a 12V input voltage. Note that the motor can only turn in one direction. The used IC can also be configured to generate a zero input voltage at a duty cycle of 50% and a positive input voltage for higher duty cycles and a negative input voltage for lower duty cycles. However, as will be explained later, it is difficult to keep track of the traveled distance when the wheels can turn both ways, therefore Trilobot's wheels are configured to travel only in positive direction.
The circuitry given in Fig. 4.6 is mounted on Trilobot to be able to control each drive-motor. The circuit is fed with the 12V power supply from the battery pack. An additional switch and power LED are included in the circuitry to be able to switch the power on and off since it is not connected to the main power switch of Trilobot.
In order to drive the motors, a PWM signal has to be generated on both the PWM inputs (PWM1 and PWM2 in Fig. 4.6) of the motor circuitry. Both PWM signals can be generated on a pin of the processor.

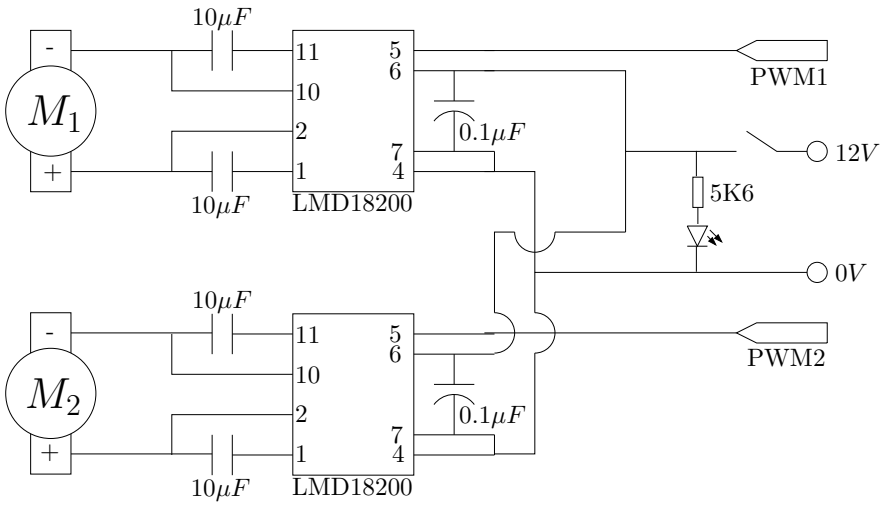


Figure 4.6: Pulse Width Modulation circuit for Trilobot's DC motor.

4.3.2 Drive-motor software

To drive the motors, two PWM signals have to be generated on two of the processor output pins. It is required that the duty-cycles of these two PWM signals can be varied independently. Preferably, the frequency of both signals has to be equal, to ensure both drive-motors react in the same way to a given duty-cycle.
Nowadays many microcontrollers have one or more configurable PWM outputs. Such microcontrollers exhibit a special data register that holds the value of the duty cycle, which can be altered during program execution. Unfortunately, the microcontroller on-board Trilobot does not have this feature, so both PWM signals have to be generated using software.
As mentioned before, a timer can be used to set a fixed frequency. For the generation of a PWM signal on an available port of the microcontroller, one of the timers is used in 8-bit auto reload mode. The timer counts from zero to 255 and then overflows back to zero in approximately 0.28ms and this time interval will be used to set the frequency of the PWM signal (which will be around 3.6kHz). The duty cycle will be stored somewhere in memory as an 8-bit value. In order to generate a PWM signal with the given frequency, a port bit has to be high for a number of machine cycles equal to the duty-cycle stored in memory. Subsequently, the same bit has to be set low for a number of machine cycles complementary to the value of the duty cycle stored in memory. For this purpose, the auto-reload function of the timer can be used.
Consider for example the generation of a 25% duty-cycle PWM signal. The value of the duty-cycle is stored in memory as 64 (which is 25% of the maximum of 256 of an 8-bit integer) as well as the 8-bit complement of that value (256-64=192). The reload value of the timer and the value of the timer itself are loaded with the complement of the duty-cycle and the port on which the PWM signal has to be generated, is set high. The counter will count from 192 to 255 and then overflows. When the timer overflows, the timer will be stopped and the reload value and the timer value will be loaded with the value of the duty cycle. Furthermore, the PWM port is set low and the timer will be started again. On the next overflow (in this case after 192 cycles) the port will be set high again and the timer value and the reload value will be loaded with the complement of the duty cycle, and so on. Fig. 4.7 shows the value of the

timer and the signal generated on the PWM port in case of a 25% duty-cycle.

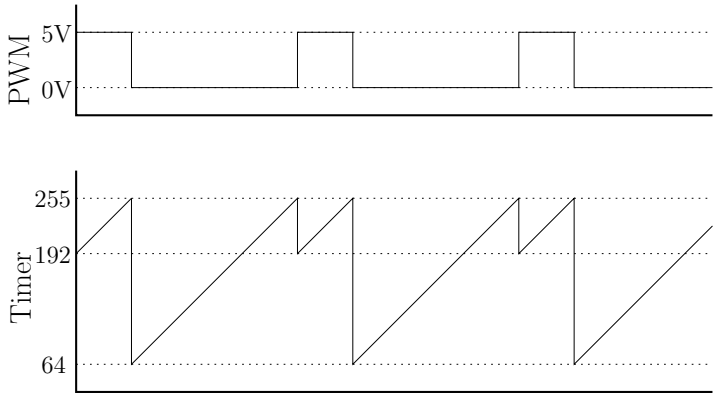
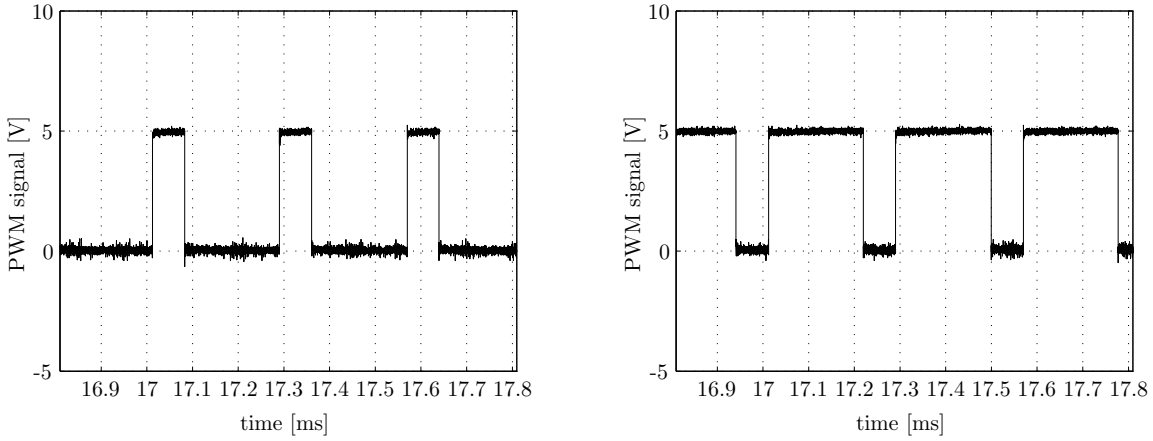


Figure 4.7: The PWM output and the value of the timer for a 25% duty-cycle PWM signal

Recall from section 4.1 that an interrupt can be triggered on timer overflow. Therefore the routine described above can easily be implemented as an interrupt routine and the generation of the PWM signal can be done completely in an interrupt based fashion. No operations in the main program loop are required.
Furthermore, the value of the duty-cycle can be changed in memory during operation and on the next timer overflow this value will be used instead of the old value. Fig. 4.8 shows two PWM signals generated on a processor pin using the routine described in the foregoing; one with a 25% duty-cycle and one with a 75% duty cycle. The interrupt routine written to perform this task is listed at address `1000h` in the code listed in Appendix F.1.

In order to generate two PWM signals, two processor pins have to be available and also two timers have to be used. On Trilobots expansion connector, two processor pins (P3.4 and P3.5) are available. These pins are not used to interface the other hardware on-board Trilobot. Furthermore, there are three timers available. However, one timer is needed to set the serial port Baud-rate as discussed before. Moreover, a timer is needed to count the encoder transitions and for this purpose also one of the free processor pins is needed. Finding another pin to generate a PWM signal on will not be a problem. The laser-pointer and the headlight mounted on the mast of Trilobot for example can be turned on and off using simple commands. Therefore, it is possible to generate PWM signals on the lines connected to these devices. Even though the output of these lines can be altered with one simple predefined command, it takes considerably more machine cycles to complete these commands than setting or clearing a processor pin directly.
With some extra code, two PWM signals can be generated using one timer. There are some issues, however, that have to be taken into account. Three different reload values have to be computed based on the two duty-cycles. This is illustrated in Fig. 4.9 where two different PWM signals, one with a 25% duty-cycle and one with a 50% duty-cycle and the corresponding value of the timer are drawn. The figure shows that three timer reloads are required to generate two PWM signals. It is straightforward to compute these reload values with some additional code. Another important issue is to keep track of which line has to be set low after the first and the second reload, because this order may vary when the duty-cycle of both PWM signals changes.
Another issue that plays a role is the frequency of the PWM signal. The generation of two



(a) 25 % duty cycle PWM signal.

(b) 75 % duty cycle PWM signal.

Figure 4.8: PWM signal generated using software with two different duty cycles.

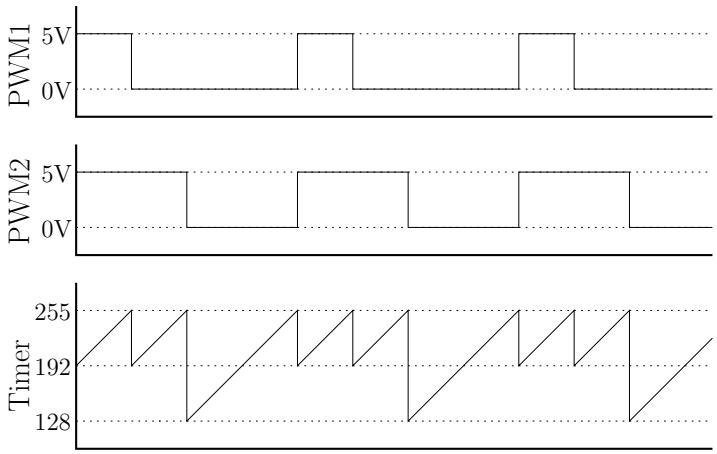
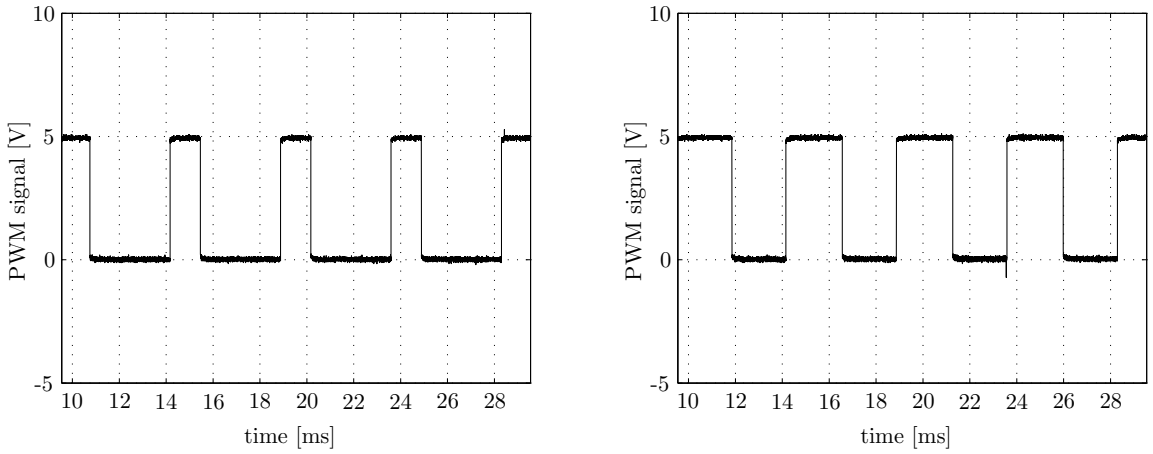


Figure 4.9: Two generated PWM outputs with one timer.

PWM signals with one timer introduces some extra computation effort, which translates into a considerable amount of extra machine cycles. Therefore, the PWM frequency discussed previously, based on a sample-time of 256 machine cycles, is too high. If all the extra instructions take for example 50 extra machine cycles, the duty cycle of the generated PWM signals could diverge about 20% from the desired duty-cycle. One of the timers however can be used in 16-bit auto-reload mode and using this timer, it is possible to raise the frequency of the PWM signals and in that way decrease the relative error in the duty-cycles due to the extra calculations. Fig. 4.10 shows a 25% and a 50% duty-cycle PWM signal generated using only one timer. Note that the frequency of the generated signal is considerably higher than that in Fig. 4.8. The interrupt routine written to generate these signals can be found in the code in Appendix F.2 at address `1000h`.



(a) 25 % duty cycle PWM signal.

(b) 50 % duty cycle PWM signal.

Figure 4.10: PWM signal generated using software with two different duty cycles.

4.3.3 Encoder readout

The encoders connected to the drive-motors are single channel incremental encoders consisting of an encoder disk and a sensor. The used encoder disks have 22 slots. The used sensor is the H21A1 optical interrupter switch by Fairchild semiconductor, which consists of an infrared emitting diode and a phototransistor. If the phototransistor detects the light source, it outputs a high signal (5V in this case). If the encoder disc interrupts the light-source, the phototransistor outputs a low (0V) signal. The coprocessor can detect these high to low transitions and in this way the displacement of the wheels can be counted with a resolution of 22 counts per revolution.
The data of the encoders can be requested from the coprocessor using a predefined command. The result, however, is the sum of the counted pulses of both encoders. These data are useless because there is no way to find out what the number of counts is for each encoder separately, which is what is required to control both drive-motors. If one encoder is disconnected from Trilobot's standard hardware, the coprocessor will return the counts of only one encoder. Using some custom software and circuitry, the disconnected encoder can be connected to the available processor pin P3.4. One of the timers can be used as an event counter to count high to low transitions on the P3.4 pin as explained in section 4.2.

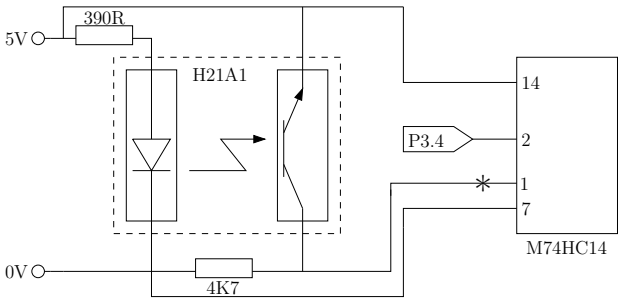
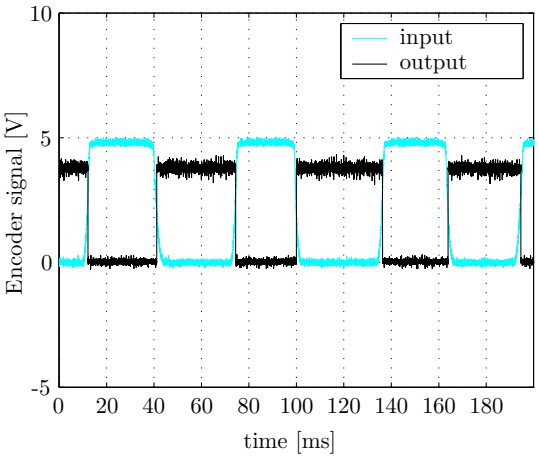
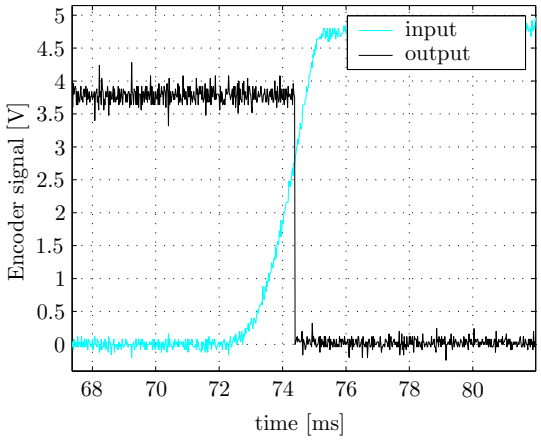


Figure 4.11: Encoder circuitry for Trilobot's encoders.



(a) Schmitt trigger input and output.



(b) Zoomed part of the left figure.

Figure 4.12: Encoder output filtered by a Schmitt inverter. The right figure shows a zoomed in part of the left figure.

The circuitry used to connect the encoder to pin P3.4 is depicted in Fig. 4.11. The H21A1 is the optical interrupter switch that is already mounted on Trilobot. The square-wave signal generated by the optical interrupter could be connected to the processor port P3.4 directly. However, the signal generated by the H21A1 is not a perfect square wave signal. The rise and fall times of the signal are considerably high and not very smooth. If it would be connected to P3.4 directly, the processor would detect these small variations in the signal as encoder counts. Therefore the device named M74HC14 in Fig. 4.6 is used. This device, called a Schmitt inverter, converts the input signal to a perfect CMOS signal. Furthermore, it inverts the input signal. When the input-signal raises above a certain threshold the Schmitt inverter outputs 0V, if the signal comes below another threshold level, it outputs 5V. Fig. 4.12 shows an output signal of the interrupter switch before filtered by the Schmitt inverter and after filtering by the Schmitt inverter which illustrates the effect of the Schmitt inverter.
The circuitry given in Fig. 4.11 is mounted to one of the encoders on-board Trilobot. Therefore, it is now possible to obtain the counts of both encoders separately.

In the software, timer 1 is configured as an event counter for port P3.4. The number of encoder counts is stored in a special register and the processor increments that register whenever a high to low transition is detected at port P3.4. The processor checks this line every instruction cycle (so with a frequency just above 900kHz). The maximum speed of the drive-wheels is 73rpm, which brings about 27 encoder counts per second, so the processor can easily count all the pulses generated by the encoder circuitry. In fact, new encoder disks with a much higher resolution could be mounted on Trilobot without effort to improve the accuracy of the position measurement. When timer 1 is used as an event counter, it can hold a 16-bit value. Note that an overflow of timer 1 occurs after 65536 counts which is after 268m. To be able to calculate the number of encoder counts effectively beyond this point, this overflow has to be registered. This can be done on-board Trilobot, but also on the external PC when it is used to control Trilobot's drive-motors.

4.3.4 Serial interfacing

The 8052 microcontroller has an on-board serial port as explained in section 4.1. The serial port makes use of an 8-bit register to store incoming bytes. As soon as a byte arrives, an interrupt can be triggered to process the incoming byte.
The Baud-rate of the serial port is the data transfer rate over the line in bits per second [11]. If no Baud-rate is configured, the UART uses the crystal frequency to clock out a data bit every machine cycle, which results in a Baud-rate of 921583 Baud.
If two devices have to communicate using a serial connection, however, both end points have to use the same connection settings and Baud-rate. Therefore, a set of standard Baud-rates is used normally in serial communication such as 1200, 2400, 9600 and 19200 Baud. To set such standard Baud-rates on the 8052, a timer can be used. A Baud-rate of 9600 for example means that 9600 bits have to be clocked out per second. If the Baud-rate is set using a timer, the Baud-rate is equal to the frequency of the timer overflow, i.e. every time a timer overflows a bit will be clocked out. In order to achieve a desired Baud-rate, the reload value of the timer can be set. The following formula can be used to compute the right reload value:

$$\text{reload value} = 256 - ((\text{crystal frequency} \;/\; 384)/\text{Baud-rate}). \tag{4.1}$$

When a Baud-rate of 9600 is required, the reload value of the timer that determines the Baud-rate has to be set to 253 when a 11.059MHz crystal is used. If for example a 12 MHz crystal is used, the resulting reload value to obtain a Baud-rate of 9600 would be approximately 252.75, which cannot be set. Therefore the initially strange frequency of 11.059MHz is used.

When the drive-motors have to be controlled over the serial line, duty-cycle information has to be sent to Trilobot. If only one motor has to be controlled, an 8-bit unsigned integer (0-255) can be used to alter the duty cycle. When the 8-bit integer arrives at the serial port, the duty cycle can be adapted according to the incoming value.
When two motors have to be driven, it has to be clear for which motor the duty-cycle information is destined. This could be solved by sending two bytes of data, one with information on the motor for which the data are meant and one byte with duty-cycle information. The latter, however, doubles the serial data traffic.
Another way to solve this problem is to use signed integers (from -128 to 127). The most significant bit of a signed 8-bit integer determines the sign of the integer given by the other 7 bits. In this way, the sign of a value determines its destination. Note that the duty-cycle was stored as an 8-bit unsigned integer. Therefore, the absolute value of the incoming data will be doubled to obtain an unsigned integer. Note that this solution divides the resolution of the PWM signal into half.
In appendix F, the user-programs written to drive both wheels and to drive one wheel are listed. In these programs the Baud-rate can be set using the dip switch on the main board of Trilobot. The following Baud-rates can be set: 1200, 2400 9600 and 19200 Baud.

When two motors have to be controlled, all the timers are in use because a timer has to be dedicated to the registration of encoder counts. In that case there is no timer available to set a fixed rate with which the microcontroller sends data to the serial port. The frequency of the PWM signals can be used to set a fixed sample rate. The only drawback of this approach is that the sample frequency can not be set to an arbitrary value but is a fraction of the PWM frequency.

In the case in which only one wheel has to be driven, an extra timer is available. Using the auto-reload option, a fixed rate can be set to send encoder data to the serial port.

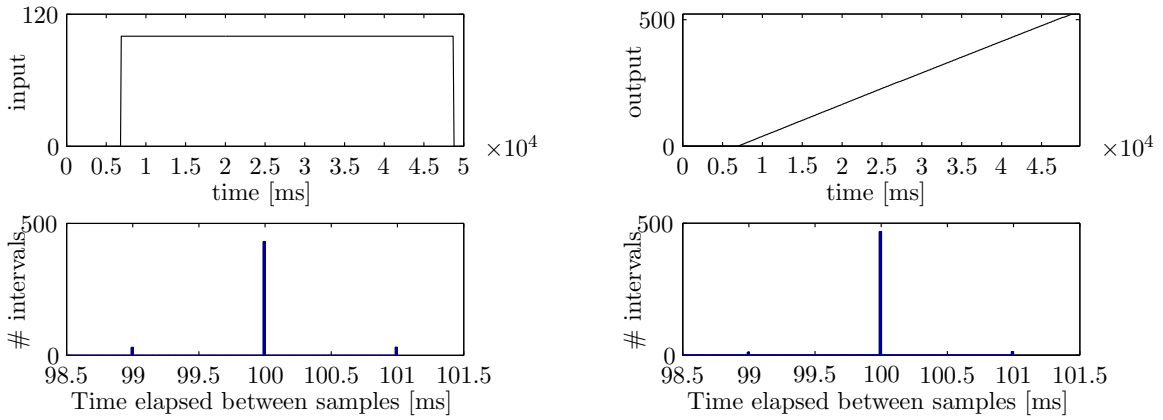
4.4 The desktop PC controller

With the user-programs discussed in the foregoing section, the drive-motors of Trilobot can be controlled from a desktop PC. Encoder information can be imported via the serial port and duty-cycle information can be sent over the same line to Trilobot to effectively change the supply voltage of the drive-motors.
The desktop PC used for this setup runs MATLAB/Simulink under Windows 2000. A Windows 2000 DLL has been written that defines a number of routines to setup the serial connection with Trilobot and to send and receive information to and from Trilobot. The DLL has been written in C++ using the Win32 Application Programming Interfaces (APIs) [19]. These routines can be invoked in Simulink using an S-function [18].
One important issue that has to be addressed in the context of this research is the timing of events such as sending and receiving data. In order to be able to measure the exact delay between the reception of data and the sending of data, a global time is necessary. In Windows, the system time can be obtained using the `GetSystemTime` API which returns the system time with a resolution of milliseconds or the `GetSystemTimeAsFileTime` API which even returns the system time with a 100 nanoseconds resolution [19]. Unfortunately, the clock interval of the windows NT family of operating systems (of which Windows2000 is part), is 10 milliseconds. So although the discussed APIs return a time with a resolution up to milliseconds, the Windows NT clock will only be updated every ten milliseconds. To obtain timing with a higher resolution, performance counters can be used. A performance counter is a high resolution hardware counter which can be used to measure brief periods of time with high precision [22]. A high resolution timer which makes use of the performance counter is included in the routines in the DLL to register the time at which sampling events occur.

The result of the approach described in this section are two Simulink blocks that can be used to control Trilobot in combination with one of the user-programs using a serial connection. One block for the single wheel case and one block for the case where both wheels have to be controlled. The user can select a desired Baud-rate and sample-time. Also, real-time information on the exact sampling instants of the controller and the instant new encoder data arrive can be gathered optionally. The code of the two DLL's and the S-functions used within Simulink is listed in Appendix D and Appendix E.

4.5 Experimental settings

For the experiments discussed in the next chapter, the single wheel setup is used. The sample rate of Trilobot is set to 10Hz. Every 0.1s the position information of the encoder of the considered wheel is sent to the desktop PC over the serial line. The used Baud-rate is 19200 bits per second. The encoder data consist of one 8-bit value. These eight bits have to be sent to the desktop PC with one start bit and two stop bits that have to be added to transfer the data using the RS-232 protocol [11]. With the used Baud-rate it takes approximately 0.57 milliseconds to transfer the eleven bits, which is 0.57 percent of the used sample-rate.



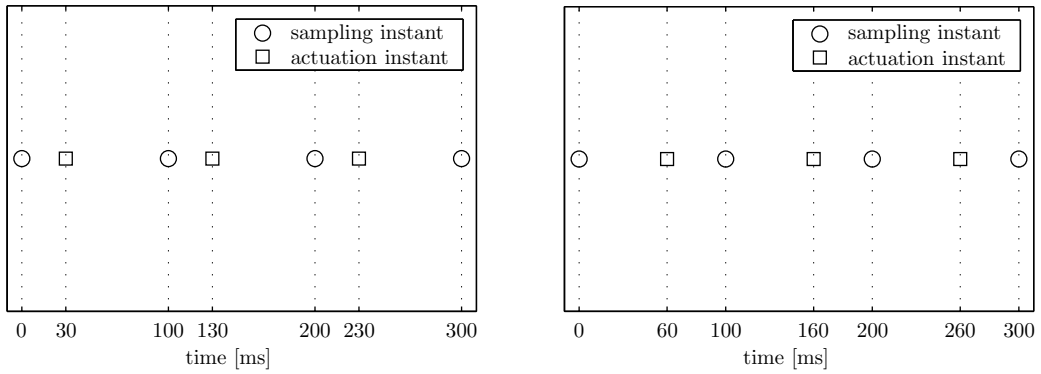
(a) Input of Trilobot.

(b) Output of Trilobot.

Figure 4.13: A constant input of 100 applied to the discussed setup. The sample rate is set to 0.1 s.

Transferring PWM data back to Trilobot takes the same amount of time. In order to be able to set the delay to an arbitrary value, the time-keeping routines described in the previous section are used to wait a specified number of milliseconds before the input computed by the controller is sent to Trilobot. When these data arrive at Trilobot, they will be processed immediately due to the event-driven nature of the actuation. The communication delays due to the serial communication can be neglected compared to the extra added delay and therefore the timing of the sampling instant and the actuation instant can be done on the desktop PC. Of course, also a smaller sample rate could be used and then the delay introduced by the serial communication is a significant percentage of the sample-time. In that case, however, it is much more difficult to control the delay in the system and setup the desired conditions. Furthermore, some type of clock synchronization is necessary to obtain a global time registration of the actions on-board Trilobot and on-board the desktop PC.
Fig. 4.13 shows the results of applying a constant input to the motor of Trilobot's drive-wheel. The left figure shows the input to Trilobot which is a constant value of 100. The lower graph gives a histogram of the elapsed time between the actuation instants. The actuation takes place when a sample is received from Trilobot after a fixed (manually set) delay. So the actuation rate is the same as the sampling-rate set on board Trilobot. The right figure shows the output of Trilobot. The lower figure again illustrates the sampling-rate set on-board Trilobot which is very accurate.

Fig. 4.14 shows the timing of the sampling instant and the actuation instant for a delay of 30ms and 60ms. As can be seen in the figure, the delay between sampling and actuation can be set very accurately in the used setup. The zero delay case cannot exactly be set because there is always a small delay due to the serial communication. Furthermore, the control input is computed within Simulink. Simulink runs at a sample-rate of 1KHz and it samples the data obtained by the routines in the DLL. Although the sample-rate of Simulink is 100 times higher than the sample-rate of Trilobot, the sampling behavior of Simulink can introduce an extra delay up to one sampling-interval, which is 1ms for a 1KHz sample-rate.



(a) 30% delay.

(b) 60% delay.

Figure 4.14: Timing of the sampling and actuation instant for a 30% and a 60% delay value.

Chapter 5

Experiments

In this chapter, the experiments done to validate the results obtained in chapter 3 are discussed. First, a linear continuous-time model will be derived for the used single wheel setup. Next, a Continuous Discrete Extended Kalman Filter is used to estimate parameters that arise in the model. A brief explanation of the used filter is given and the estimation process using measurements of the setup is discussed. The velocity of the wheel is obtained using a state estimator during the experiments and a brief explanation of the used estimator is given. Next, the numerical results as obtained in chapter 3 will be given for the derived model of the setup in terms of a stability bound for a position feedback and a full state feedback. Both cases are validated using experiments. This chapter ends with a brief discussion on the obtained results.

5.1 Model of the setup

The experiments to validate the theoretical results discussed in chapter 3 will be conducted using the single wheel configuration of Trilobot discussed in chapter 4. In order to compare the experimental results to the results obtained analytically, a linear continuous-time model of the setup has to be derived that can be used to perform the analysis carried out in chapter 3.
A schematic representation of one of Trilobot's wheels is given in Fig. 5.1. The following equations describe the dynamics of the wheel:

$$\hat{J}\ddot{\theta} + \hat{b}\dot{\theta} = T = c_m u, \tag{5.1}$$

where $\hat{J}$ is the inertia of the wheel, $\hat{b}$ is a viscous damping parameter, T is the applied torque and c_m is the motor-constant. θ is the rotation of the wheel expressed in encoder counts and u is the input (i.e. PWM information sent to Trilobot). The input lies between 0 and 255 which corresponds to an input voltage between 0 and 12V.
In state-space notation (5.1) can be written as:

$$\begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & \frac{-b}{J} \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{J} \end{bmatrix} u. \tag{5.2}$$

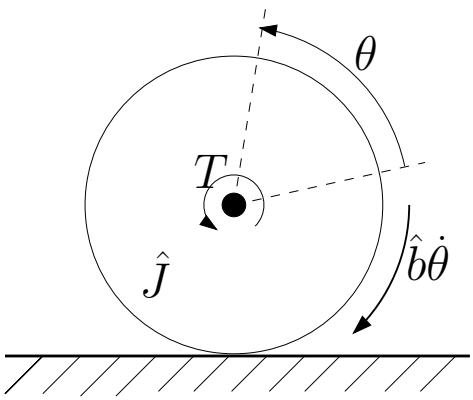


Figure 5.1: Schematic representation of Trilobot's wheel.

Note that in this notation $J = \frac{\hat{J}}{c_m}$ and $b = \frac{\hat{b}}{c_m}$.
In order to be able to compare the experimental results to the analytical results, accurate values for the parameters J and b have to be identified or estimated.

5.2 Identification of the system parameters

The parameters arising in the state-space system equation in (5.2) will be estimated using a Continuous Discrete Extended Kalman Filter. In general, a Kalman filter is an estimator that can be used to reconstruct the states of a system, minimizing the variance of the difference between the actual state and the estimated state.
An Extended Kalman Filter can be used if the process to be estimated and/or the measurement relation to the process is nonlinear. The Jacobians of the process to be estimated and the measurement function are used to linearize around the current estimate.
Because of the limited sample-time that can be obtained with the used setup, a Continuous-Discrete Extended Kalman filter (CDEKF) is used. With a CDEKF, a continuous-time process can be estimated using discrete time measurements.
In the following sub-section, a brief survey of the CDEKF will be given. An extensive discussion on the CDEKF can be found in [26].

5.2.1 The Continuous Discrete Extended Kalman Filter

The continuous-time system of which the state has to be estimated can be described by the following differential equation:

$$\dot{\underline{x}}(t) = f(\underline{x}(t), t) + \underline{w}(t) \qquad \underline{w}(t) \sim N(\underline{0}, Q(t)). \tag{5.3}$$

In this equation f is a non-linear function of the continuous-time state $\underline{x}(t)$. $\underline{w}(t)$ is a zero-mean Gaussian white noise with spectral density $Q(t)$ which represents the errors introduced by the imperfect model of the physical system that is considered.
The state $\underline{x}(t)$ will be estimated using discrete measurements of the continuous-time system output. The measurement equation is given by:

$$\underline{z}_k = \underline{h}_k(\underline{x}(t_k)) + \underline{v}_k \qquad \underline{v}_k \sim N(\underline{0}, R_k), \tag{5.4}$$

where $\underline{z}_k$ is the kth discrete measurement. The time t at the kth sampling instant is denoted by t_k. $\underline{h}_k$ depends on the state $\underline{x}(t_k)$ at each sampling time. The sequence $\{\underline{v}_k\}$ is a white random sequence of zero-mean Gaussian random variables with associated covariance matrices $\{R_k\}$ which represents measurement errors. The expected value $E[\underline{w}(t)\ \underline{v}_k^T] = 0$ which means that $\underline{w}(t)$ and $\underline{v}_k$ are uncorrelated.
For the initial condition $\underline{x}(0) \sim N(\hat{\underline{x}}_0, P_0)$, where $\hat{\underline{x}}_0$ is the initial condition of the discrete estimated state $\hat{\underline{x}}_k$ and P_0 is the initial condition of the error covariance P_k, the propagation of the state estimate $\hat{\underline{x}}(t)$ is given by:

$$\dot{\hat{\underline{x}}}(t) = f(\hat{\underline{x}}(t), t). \tag{5.5}$$

As mentioned before, the filter is based on minimizing the covariance of the estimated state. The error covariance matrix of the estimate $P(t)$ is defined as:

$$P(t) \triangleq E\left[[\hat{\underline{x}}(t) - \underline{x}(t)][\hat{\underline{x}}(t) - \underline{x}(t)]^T\right]. \tag{5.6}$$

An approximation of the propagation of the error covariance in the time intervals between the discrete measurements based on the Jacobian matrix of f in (5.3), is given by the following Ricatti equation:

$$\dot{P}(t) = F(\hat{\underline{x}}(t), t)P(t) + P(t)F^T(\hat{\underline{x}}(t), t) + Q(t), \tag{5.7}$$

in which the Jacobian $F(\hat{\underline{x}}(t), t)$ of f in (5.3) is given by:

$$F(\hat{\underline{x}}(t), t) \quad = \quad \left.\frac{\partial f(\underline{x}(t), t)}{\partial \underline{x}(t)}\right|_{\underline{x}(t)=\hat{\underline{x}}(t)}. \tag{5.8}$$

Using the measurements, an update for the state estimate can be calculated as follows:

$$\hat{\underline{x}}_k(+) = \hat{\underline{x}}_k(-) + K_k[\underline{z}_k - h_k(\hat{\underline{x}}_k(-))]. \tag{5.9}$$

In this equation $\hat{\underline{x}}_k(-)$ is the state estimate prior to the update and $\hat{\underline{x}}_k(+)$ is the state estimate after the update. The matrix K_k is the Kalman gain matrix at time t_k which is given by:

$$K_k = P_k(-)H_k^T(\hat{\underline{x}}_k(-))\left[H_k(\hat{\underline{x}}_k(-))P_k(-)H_k^T(\hat{\underline{x}}_k(-)) + R_k\right]^{-1}. \tag{5.10}$$

The update of the error covariance matrix P_k at time t_k is given by:

$$P_k(+) = [I - K_k H_k(\hat{\underline{x}}_k(-))]P_k(-), \tag{5.11}$$

where $P_k(-)$ and $P_k(+)$ denote the error covariance matrix P_k at time t_k before and after the update, respectively. $H_k(\hat{\underline{x}}_k(-))$ is the Jacobian matrix of $\underline{h}_k$ in (5.4) which is given by:

$$H_k(\hat{\underline{x}}_k(-)) \quad = \quad \left.\frac{\partial h_k(\underline{x}(t_k))}{\partial \underline{x}(t_k)}\right|_{\underline{x}(t_k)=\hat{\underline{x}}_k(-)}. \tag{5.12}$$

5.2.2 Estimation of the parameters

The CDEKF is used to estimate the states and the parameters b and J^{-1} of the system in (5.2). The augmented state in (5.3) is therefore given by $\underline{x}(t) = [\theta \;\; \dot{\theta} \;\; J^{-1} \;\; b]^T$. The system equation given in (5.3) applied to the wheel model is given by:

$$\underline{\dot{x}}(t) = \begin{bmatrix} \dot{\theta} \\ \ddot{\theta} \\ \dot{J}^{-1} \\ \dot{b} \end{bmatrix} = \begin{bmatrix} \dot{\theta} \\ -bJ^{-1}\dot{\theta} + J^{-1}u \\ 0 \\ 0 \end{bmatrix} + \underline{w}(t). \tag{5.13}$$

The measurement equation in (5.4) for the wheel model is given by:

$$\underline{z}_k = \theta(t_k) = [1 \;\; 0 \;\; 0 \;\; 0]\underline{x}(t_k) + \underline{v}_k. \tag{5.14}$$

Using these system equations, the Jacobian matrices given in (5.8) and (5.12) can be computed, which leads to the following Jacobian matrices:

$$F(\underline{\hat{x}}(t), t) = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -bJ^{-1} & -b\dot{\theta} + u & -J^{-1}\dot{\theta} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{5.15}$$

$$H_k(\underline{\hat{x}}_k(-)) = [1 \;\; 0 \;\; 0 \;\; 0]\,. \tag{5.16}$$

The estimation process is performed off-line. This means that first an input will be applied to the system and the output data will be measured. These data will be used to do the actual estimation afterwards.
Initial values have to be ascribed to the state vector $\underline{x}(t)$, the error covariance matrix $P(t)$, the spectral density matrix $Q(t)$ and the measurement covariance R_k. The initial value of θ and $\dot{\theta}$ are known a priori because the estimation process is performed off-line. The initial values of the parameter b and J^{-1} are difficult to predict and therefore an initial condition has to be chosen based on engineering insights: $b_0 = 5 \; N(counts)s = 8.26 * 10^{-4} \; Nms$ and $J^{-1} = 0.5 \; (kg(counts)^2)^{-1}$ this corresponds to a inertia of $J = 3.3 * 10^{-4} \; kgm^2$. Note that in the used setup, the measured output is the number of encoder counts. One encoder count corresponds to a traveled distance of approximately 12.85 mm.
The initial error covariance matrix P_0 characterizes the uncertainty on the initial states $\underline{x}_0$. The matrix P_0 is taken to be:

$$P_0 = \begin{bmatrix} \text{1e-12} & 0 & 0 & 0 \\ 0 & \text{1e-12} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \text{1e-1} \end{bmatrix}. \tag{5.17}$$

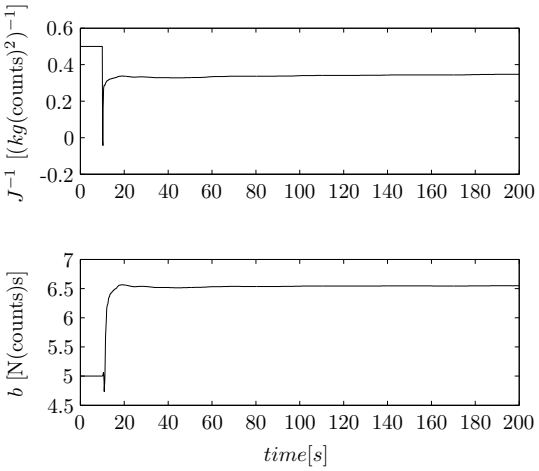
The states are assumed to be mutually independent, so the covariances in P_0 (i.e. the non-diagonal elements) are taken zero. The variance on θ and $\dot{\theta}$ can be very small because the initial value of these states is known exactly, therefore these variances are taken to be $1e-12$. The initial value of the two remaining states b and J^{-1} are not very accurate and therefore an initial variance of about 20 percent of the initial guess is ascribed to these parameters.

The spectral density matrix $Q(t)$ is used to define model uncertainties. The modeling errors in the model equation given in (5.13) are also assumed to be mutually independent, so also Q_0 is a diagonal matrix. Q_0 is taken to be:

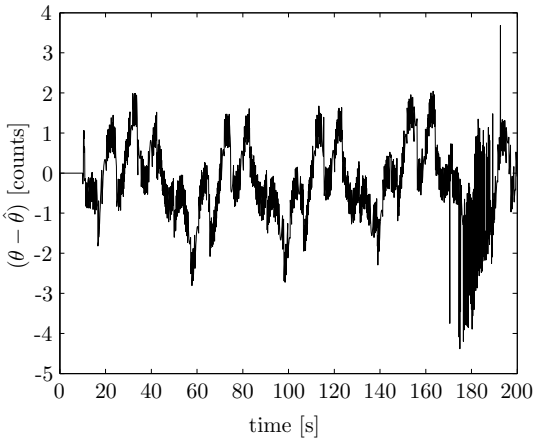
$$Q_0 = \begin{bmatrix} 1\text{e-}12 & 0 & 0 & 0 \\ 0 & 1\text{e-}5 & 0 & 0 \\ 0 & 0 & 1\text{e-}12 & 0 \\ 0 & 0 & 0 & 1\text{e-}12 \end{bmatrix}. \tag{5.18}$$

To model the drive-wheel, a simple linear model is used. Nonlinearities such as dry friction are omitted which makes the model inaccurate. The modeling errors appear in the second equation in (5.13), therefore the second diagonal element of Q_0 is set to $1e-5$. Under the reasonable assumption that the parameters b and J^{-1} are constant in time, the other three diagonal terms are set to $1e-12$.
Because the considered system has only one output, the number of encoder counts, the measurement variance is a scalar. The maximum error in the measurement that can occur is one encoder count. Therefore $R_0 = 1$.

The input signal used to capture output data is the sum of two sine-waves. It is important to notice that the input signal is chosen such that the wheel constantly moves in order to avoid sticking due to (nonlinear) dry friction.
Fig. 5.2(a) shows the convergence of the parameters b and J^{-1}. The viscous damping parameter b converges to a value of 6.5 $N(counts)s$, which corresponds to $1.1 * 10^{-3}$ Nms and the inverse of the inertia, J^{-1}, converges to 0.35 $(kg(counts)^2)^{-1}$ which corresponds to an inertia of $J = 4.76 * 10^{-4}$ kgm^2.
Fig. 5.2(b) shows the error between the measured output and the estimated output. Around 180 s, an enlargement of the error between the estimated and the measured output can be seen due to a disturbance in the serial communication. However, this does not have a large influence on the estimation of the parameters b and J^{-1}.



(a) Convergence of b and J^{-1}.



(b) Difference between the estimated and measured output $(\theta - \hat{\theta})$.

Figure 5.2: Results of the estimation process.

5.3 Reconstruction of the velocity

To apply a full state feedback, the velocity has to be known. Because only the position of the wheel can be measured using the encoders, the velocity of the wheel has to be computed using the encoder information. The analysis in chapter 3 is based on the assumption that the continuous-time state is measured and sampled. In order to provide a smooth velocity signal, a discrete state estimator is used. The equation of the estimator is given by [8]:

$$\underline{\hat{x}}(k+1) = \Phi\underline{\hat{x}}(k) + \Gamma u(k) + L[y(k) - C\underline{\hat{x}}(k)], \tag{5.19}$$

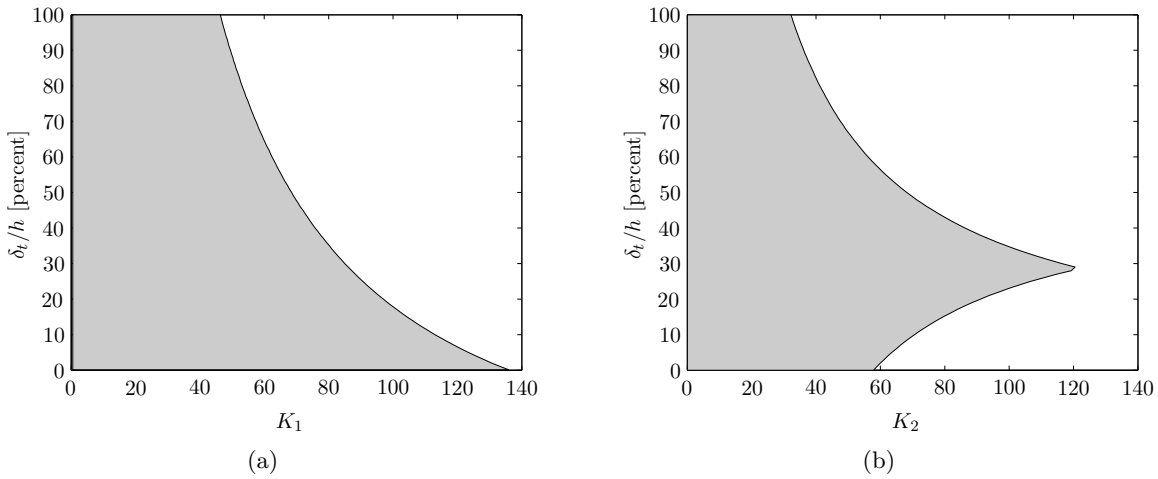
where Φ and Γ are the result of the zero-order-hold discretization of the continuous-time model of the single wheel setup. The difference equation that describes the dynamics of the error $\underline{e}(k) = (\underline{x}(k) - \underline{\hat{x}}(k))$ is given by:

$$\underline{e}(k+1) = [\Phi - LC]\,\underline{e}(k). \tag{5.20}$$

If the error dynamics represents an asymptotically stable system, the error will converge to zero. The speed of convergence can be influenced by choosing the roots of $[\Phi - LC]$ sufficiently fast.
Because of errors in the model and measurement disturbances, $\underline{\hat{x}}(k)$ will not exactly be equal to $\underline{x}(k)$. However, the estimator gains $L = [L_1 \quad L_2]^T$ can be used to tune the estimator so that the estimator is stable and the error is acceptably small. If $L = [1 \quad 0.5]^T$ the roots of $[\Phi - LC]$ are approximately $[0.06 \quad 0.74]$ and this leads to a satisfying estimation of the state.

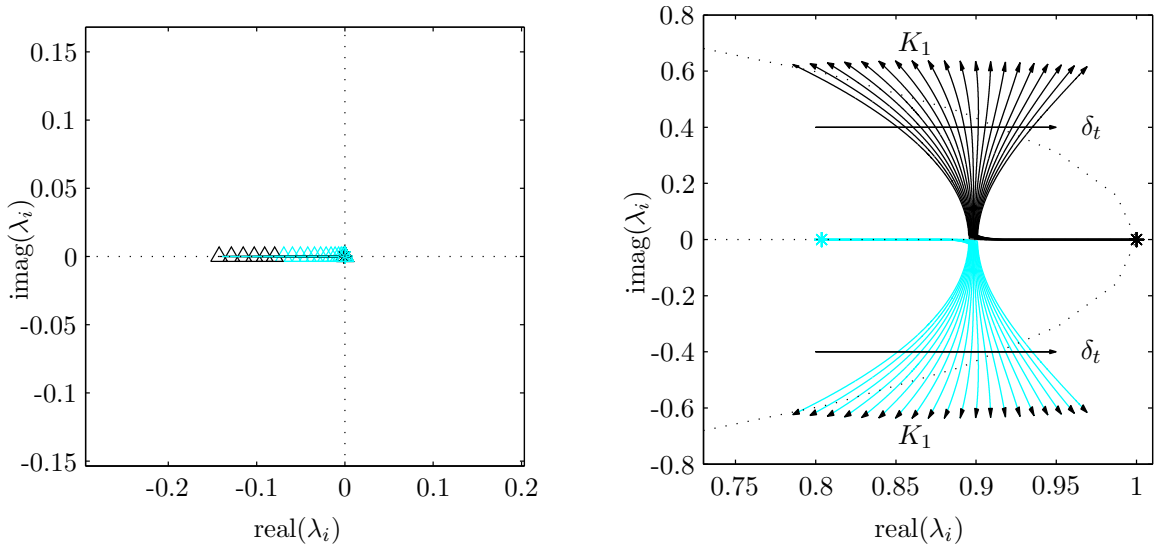
5.4 Numerical results using the estimated model

Fig. 5.3(a) shows the result of the numerical stability analysis as discussed in chapter 3 using the wheel model given in (5.2) and a feedback $u = -K_1\theta$. The used sample-time is $h = 0.1$ s. The root loci for this case are depicted in Fig. 5.4. Note that the x-axis, the y-axis and the unit-circle are represented by the dotted lines.
In these plots K_1 is varied between 0 and 145 and the delay δ_t is varied between 0 and 100 percent of the sample-time. In the left figure, the first two of the four root loci are plotted. The start of each root locus is marked with an asterisk and the end is marked with a triangle. As can be seen in the figure, the first two roots do not have an effect on the stability of the system since they lie in the unit-circle for all values of K_1. In the right figure, the root locus of the other two roots is shown. The arrows indicate the directions of increasing δ_t and K_1. An asterisk marks the start of the root locus. One of the two roots always starts on the unit-circle. This is the first point of the root locus where $K_1 = 0$ and also in this case the system can only be stabilized for $K_1 > 0$. The stable region in Fig. 5.3(a) is completely determined by the two roots shown in the right figure of Fig. 5.4.
Due to the large damping in the system, the system can be stabilized using only position feedback with a gain up to 140. As expected for the position feedback, the region of K_1 for which the system is stable decreases with increasing delay δ_t.



(a) (b)

Figure 5.3: Stability of the NCS when using the model of the wheel of Trilobot. The shaded area indicates the stable region. $h = 0.1$ s.



(a) (b)

Figure 5.4: The root loci for the position $u(k) = -K_1 x(k)$ feedback for varying K_1 and increasing delay value δ_t. $K_2 = 0$ and $h = 0.1$ s.

Fig. 5.3(b) gives the stable region for the full state feedback $u = -K_1\theta - K_2\dot{\theta}$ with $K_1 = 1$ and K_2 varying between 0 and 145.
As expected the same phenomenon occurs when applying a velocity feedback to the two dimensional system, see section 3.3. For a delay of approximately 30%, the set of K_2 values for which the system is stable, is the largest. Fig. 5.5 shows the root loci for this case. In the left figure again the start of each root locus is marked with an asterisk and the end with a triangle. In the right figure, the arrows indicate the direction of increasing K_2 and δ_t. The same pattern can be recognized as in the two-dimensional example in section 3.3 that results

in the stable area given in Fig. 5.3(b).

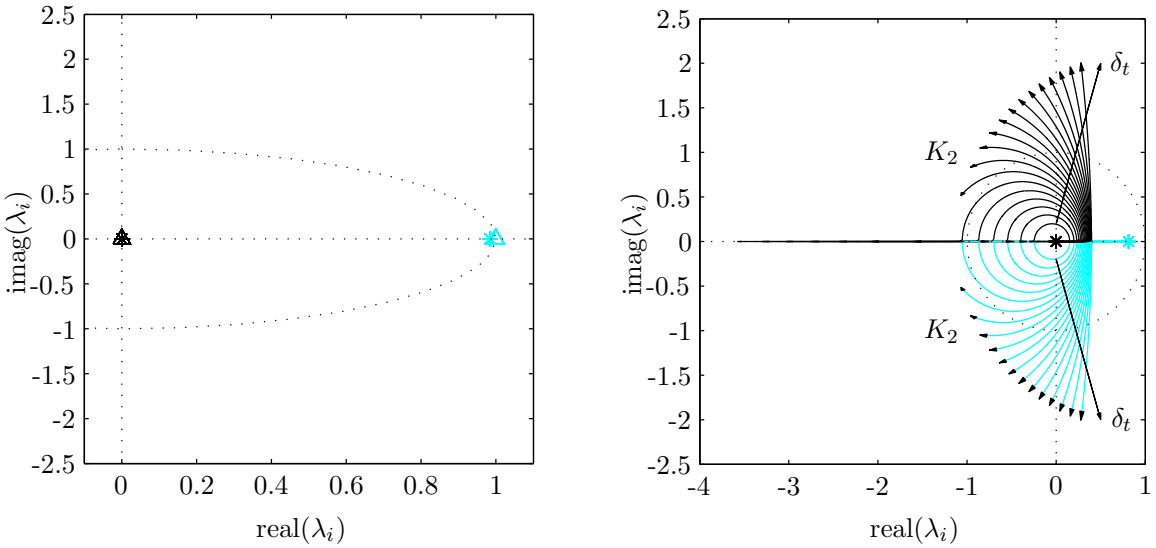


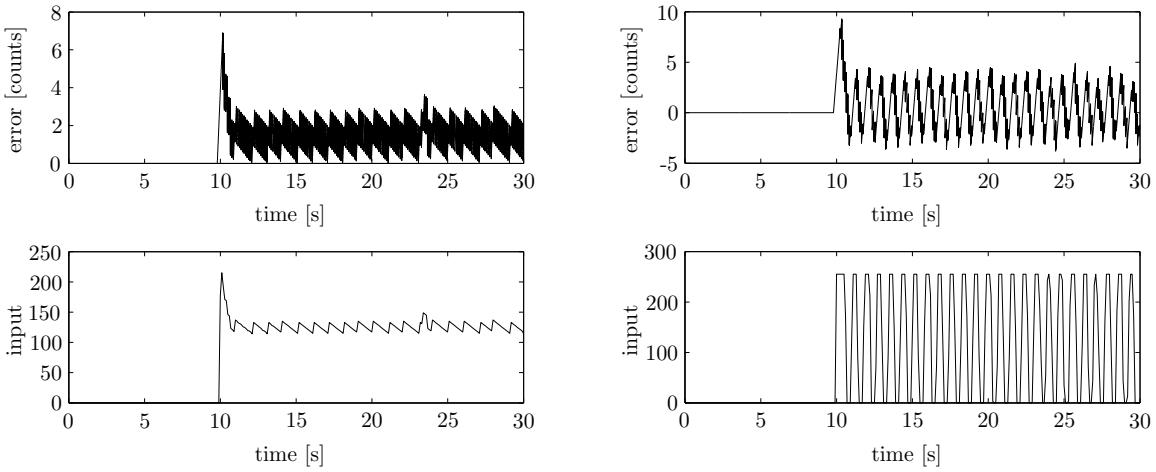
Figure 5.5: The root loci for the full state feedback $u(k) = -K_1 x - K_2 \dot{x}$ for varying K_2 and increasing delay value δ_t. $K_1 = 1$ and $h = 0.1$ s.

5.5 Experimental results

To validate the analytical and numerical results obtained in the foregoing, some tracking experiments will be performed using the single wheel setup discussed in chapter 4 and of which the numerical results are given in the previous section.
One drawback is that the control input is only defined between 0 and 255, where an input of 255 corresponds to a 12V input. If the control input exceeds these limits, the input saturates and the experiments are not useful to validate the obtained results. As a consequence, a position feedback controller $u(k) = -K\theta(k)$ would saturate if $\theta(k) > 0$, i.e. positive errors cannot be compensated because the wheel can only turn in one direction. Therefore a stabilization around a reference velocity $\dot{\theta}_{ref}$ is performed. The position reference $\theta_{ref}(t)$ is therefore a linear increasing displacement. The following controller is used:

$$u(k) = u_{ref} + K(\underline{\theta}_{ref}(k) - \underline{\theta}(k)), \tag{5.21}$$

where $K = [K_1 \quad K_2]$, $\underline{\theta}_{ref} = [\theta_{ref} \quad \dot{\theta}_{ref}]^T$ and $\underline{\theta} = [\theta \quad \dot{\theta}]^T$. The reference input $u_{ref} = b\dot{\theta}_{ref}$ in order to shift the equilibrium point to $\underline{\theta} = \underline{\theta}_{ref}$. u_{ref} is fixed to 100 which gives a reference velocity $\dot{\theta}_{ref} \approx 15$ counts/s. The sampling-time is fixed to $h = 0.1$ s and the total delay in the system can be varied from 0.005 s to 0.09 s as explained in chapter 4.
In order to validate the numerical results in Fig. 5.3, the experiment described above is performed for several delay values δ_t, trying to find the stability border of the controlled system by increasing the gain K_1 or K_2.
Obviously, the exact stability border is not easy to pinpoint. The first reason for that is that the control input cannot blow up because the input saturates at 255.



(a) $\delta_t = 0.04$ s, $K_1 = 20$.

(b) $\delta_t = 0.04$ s, $K_1 = 100$.

Figure 5.6: Experimental results for $\delta_t = 0.04s$.The upper figure gives the tracking error, the lower figure gives the control input. $K_2 = 1$ and $h = 0.1$ s.

Consider for example Fig. 5.6. The upper graph of both figures shows the tracking error $\theta_{ref}(k) - \theta(k)$ in encoder count and the lower graphs show the control input $u(k)$. In the left figure, the feedback gain $K_1 = 20$ and in the right figure $K_1 = 100$. $K_2 = 0$, $\delta_t = 0.04$ s and $h = 0.1$ s for both figures. After ten seconds, a step in the reference velocity $\dot{\theta}_{ref}$ is applied. As can be seen in the right figure, the control input saturates as soon as the change in the reference signal occurs. The control input switches from zero to 255 and back. Although this type of on/off control does not lead to an ever increasing tracking error, as can be seen in the upper graph of the right figure, this behavior can be considered unstable.

Fig. 5.7 shows the ranges where the control input starts to saturate and constantly switches between zero and 255 for varying δ_t. The results are plotted over the numerically obtained stability border. Fig. 5.7(a) gives the results for the position feedback for varying K_1. In appendix A.1 the tracking error and the input sent to Trilobot is given for three experiments for settings of K_1 before and after the stability border. As can be seen the experimentally obtained stable ranges for different values of δ_t resemble the computed stability bound. The range of K_1 for which the controlled system is stable, decreases as the delay in the loop increases. The maximum value of K_1 for each delay value is slightly below the computed values. This can be attributed to the inaccuracy of the model and disturbances on the system and measurements during the experiments and the saturation of the controller. Fig. 5.7(b) shows the regions where the input starts to saturate and constantly switches between zero and full input for the full state feedback case for varying values of δ_t. Note that $K_1 = 1$ for all experiments. To get a better resolution, measurements with a 15% delay are added. The experiments do not quite resemble the numerically obtained results. However, the same trend can clearly be seen in the experimental and numerical results. Several reasons can be given why the experimental results deviate from the numerically obtained results. The limited accuracy of the velocity obtained by the state observer plays a role in this. Furthermore, model errors and measurement disturbances again contribute to the fact that measurements and numerical results differ.
The largest set of K_2 for which the system is stable however, is measured when $\delta_t = 0.02$.

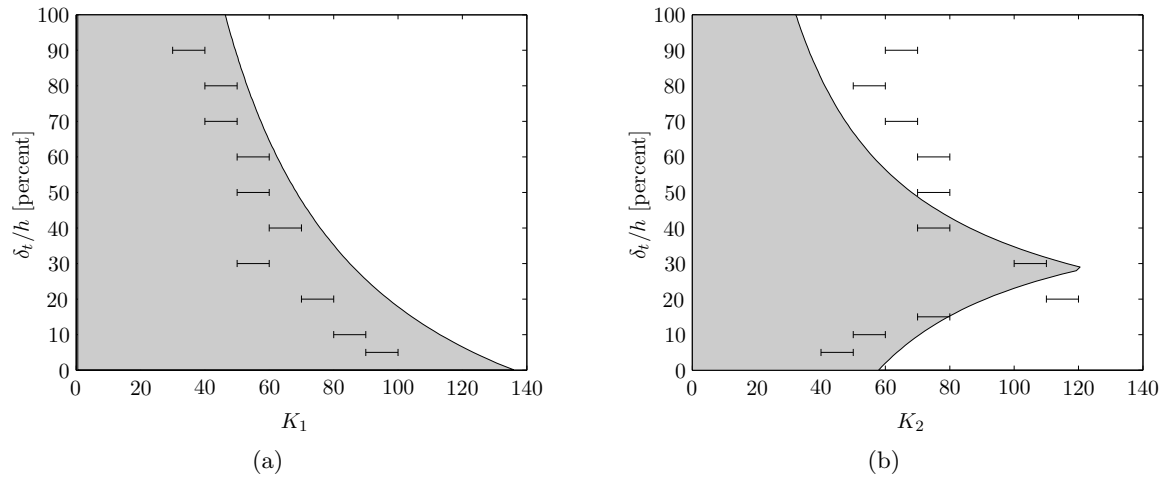


Figure 5.7: Results of the experiments. The lines indicate the region where the system starts to get unstable in the experiments. The shaded area indicates the stable region obtained numerically.

And although the value of δ_t^* (the delay value that gives the largest set of K_2 for which the system is stable) is shifted down a bit, the phenomenon that a small amount of delay yields a larger stable region can be clearly observed.

5.6 Discussion

In this chapter, the validation of the numerically obtained results using experiments is discussed. A Continuous-Discrete Extended Kalman Filter is used to identify the parameters that arise in the linear continuous-time model of the single wheel setup. Because negative inputs are not defined in the used setup a stabilization around a constant reference velocity has been performed to analyze the stability of the controlled system for varying control gains and varying delay in the control loop. Although the numerical results and the experimental results do not match perfectly, the phenomena observed during the theoretical analysis can clearly be identified in the experiments.

Chapter 6

Conclusions and recommendations

In this chapter, conclusions will be drawn as a result of the study presented in this thesis. At the end of this chapter, recommendations for future research will be given.

6.1 Conclusions

In this thesis, the issues arising in the control of networked systems have been explored. At first, the parameters that play a role in the control of networked systems are identified. Secondly, the influence of these parameters on the stability of an NCS under certain assumptions is investigated. A third and last topic discussed in this thesis is the Trilobot mobile robot. This has been adapted such that it can be used in this research and for use in control related research in general. Conclusions will be drawn concerning these three main topics in this section consecutively.

6.1.1 Sampling and delay in an NCS

Although the advantages of the use of networks in control systems are clearly present (e.g. the increased flexibility and ease of maintenance of such setups), there are some major drawbacks that have to be dealt with, when using network connections to mutually attach different parts of a control system.
A delay is introduced in the control-loop which is generally undesirable. The characteristics of this delay are strongly dependent on the type of network and the number of nodes that make use of this network. Furthermore, these network delays are often non-deterministic due to the access methods used to control the network traffic.

Due to network transfers, the sample-rate of an NCS is limited. In a digital control system normally a sample-rate can be chosen based on the desired bandwidth of the closed loop system and the limitations of the hardware. Generally, a higher sample-rate leads to a better performance of the controlled system. When the sample-rate increases in an NCS, the network load increases and as a consequence the delay in the control loop increases. Therefore a sensible choice for the used sample-rate has to be made within the limitations imposed by the network.

The characteristics of the delay in the control-loop can be influenced by the choice of the sample-rate, but also by the choice of sampling-type of the devices in an NCS. A distinction can be made between event-driven and time-driven devices.
Event-driven devices process incoming data immediately and therefore no extra delay is introduced by the sampling process itself. The disadvantage of event-driven devices is that the sample-time is not fixed but can vary every sampling-instant. Therefore, it is difficult to analyze systems with event-driven devices. It is possible to make use of time-stamps of signals to compute the actual delay of a sample and reckon with the delay in the used control strategy. This however, requires extra time-stamp data to be sent over the network which again induces an increase in the delay. Furthermore, some clock synchronization between the devices connected via the network is required.
When making use of time-driven devices, extra delay is introduced in the control loop. If the upper-bound of the network delay is known, however, a sensible choice for the sample-rate and the sampling strategy can result in a constant delay in the control loop. Although this delay is larger than the delay induced by the network, it makes the analysis of the system much more straightforward.
The introduction of a time-skew between sampling instants can be helpful to minimize the extra delay introduced by the sampling process. To assure such a time-skew also some type of clock synchronization is required.

6.1.2 Stability analysis of an NCS with constant network delays

A discrete-time NCS model for constant network delays has been derived under the assumption that the total delay in the loop is smaller than the sample-time. In this model, the sample-time and the network delay arise as parameters. The model provides a mathematical basis to analyze the influence of delay and sample-time on the stability of an NCS. The analysis of an example with one-dimensional continuous-time plant dynamics provided insight in the influence of sample-time and delay on the controlled system. The one-dimensional case allows the use of several analytic techniques to perform stability analysis such as Jury's test and frequency domain techniques.
When using a system with two dimensional continuous-time plant dynamics, which has more physical meaning, the discrete-time NCS model becomes more complex and the analytic techniques mentioned previously cannot be used. Eigenvalue analysis however proved to be helpful to analyze the stability in this case and encountered phenomena can also be identified in the two dimensional case.

Delay in a control loop is well known for degrading the performance of a control system and affect its stability. Furthermore, an increase of the sample-time results in decreasing performance and smaller stability regions. An interesting observation however is that, when applying a feedback $u = -Kx^{(n-1)}$ to an n-dimensional system, the region of K for which the system is stable increases with increasing delay up to a certain delay value where the stable region is the largest. Note that $x^{(n-1)}$ is the $(n-1)$th time derivative of x.

An experimental rotating wheel setup has been modified to make it suitable to conduct experiments to validate the results obtained analytically and through simulation. The delay in the control loop and the sample-time can be set very accurately. A linear continuous-time model of the setup has been stated and the model parameters have been successfully obtained using a Continuous-Discrete Extended Kalman Filter. Using this setup, several experiments

have been performed to validate the result obtained by means of simulation. During the experiments, a state estimator has been used to effectively estimate the angular velocity of the rotating wheel.
Although the experimental results do not match perfectly with the numerical results, the trends that can be observed are the same in both cases. Several reasons can be pointed out why the experiments do not exactly match the result obtained numerically, e.g. model and measurement errors, saturation of the control input and inaccuracies in the estimation of the velocity. The obtained experimental results however, illustrate that the phenomena encountered during the theoretical analysis can be reproduced in an experimental setting.

6.1.3 Trilobot as experimental setup

The Trilobot mobile robot has been adapted to make it suitable to conduct experiments in favor of the research covered in this thesis. Moreover, the resulting setup can be used for numerous other types of experiments such as tracking experiments and embedded control. The drive-motors of Trilobot can be controlled separately from a desktop PC using a wired or wireless serial connection. Although the setup can be very useful, it has some disadvantages that are inherent to the way of operation of Trilobot and the limitations of the on-board hardware. The following list gives a few disadvantages that need to be considered when one wants to use Trilobot as an experimental setup:

- Due to hardware limitations, the sample-rate that can be achieved with the setup is limited to about 50Hz.
- If both drive-motors have to be controlled, the sample-rate cannot be set as accurately as in the single-wheel setup because there is no timer available anymore to generate an accurate sampling-time.
- Modifying the software of the setup requires knowledge of the 8052 core microprocessor and the Assembly language to program it. Although the instruction set is not very extensive, it can be a time-consuming task to get the experience required to effectively write user programs.
- Other sensors can be used, but that increases the loop-time of the user-program and consequently the maximum sample-rate that can be obtained decreases.

Although these disadvantages can be discouraging to use Trilobot for experiments, there also are some advantages that makes Trilobot a very instructive setup:

- Because of the embedded controller on-board Trilobot it is very useful to get a thorough understanding of the consequences and limitations of an embedded implementation of control related tasks. After all, numerous examples of such embedded control systems can be found in industry.
- Trilobot can be controlled via a desktop PC using MATLAB/Simulink and a wireless serial connection that is readily available. And although this comes at the cost of a limited sample-rate, it can be very useful to be not dependent on the length of cables when conducting experiments.

- Trilobot offers much more features than the two drive-motor/encoder combinations. So also numerous kinds of Artificial Intelligence experiments can be performed using for example the sonar and light sensors. Note that for these kinds of experiments a sample-rate between 10 and 50Hz is usually sufficient.

6.2 Recommendations

In this thesis the basics of the field of NCSs have been explored, and the stability of an NCS with a basic state feedback has been analyzed under certain assumption. The field of Networked Control Systems stretches far beyond the scope of this thesis. An endless list of other interesting research topics concerning NCSs can be given, for example:

- Different delay models can be used, for example stochastic delay models, and also different assumptions concerning the delay can be used such as delays longer than the sample-time. Furthermore it can be useful to investigate the influence of data-loss on the stability of the controlled system.
- In this thesis a linear discrete-time model has been used to analyze the stability of an NCS. Many other modeling techniques are available to model an NCS, such as non-linear and hybrid techniques. The use of such modeling techniques can lead to improved control of NCSs.
- In this thesis only basic state feedback has been addressed, but one can think of many control strategies to cope with the network delays in an NCS, such as optimal control or control techniques for hybrid systems.

Furthermore, the results obtained in this thesis using numerical analysis have been validated using a setup where all parameters can be set very accurately, which creates a close to ideal environment. It would be interesting to conduct experiments with some system controlled over a real random access network connection that is shared by several other devices, especially because such experimental validation of obtained results is rare in NCS research literature. Using such a setup, it would be of interest to answer the following research questions:

- If an upper-bound of the delay that occurs due to the network transfer can be found, is it possible in practice to make this delay constant by an appropriate choice of sample-rate and time-skew?
- Can the phenomenon observed in the analysis as well as in the experiments be used in practice to improve the characteristics in a real networked setting as described above?

The expansions made on the robot during the research covered in this thesis are suitable to conduct experiments using both drive-motors. However, for this research it is only used to drive one of the two motors. Therefore, the following recommendations concerning Trilobot can be given:

- With the two drive-motors and all the other sensors on-board, Trilobot can be used for tracking experiments. As mentioned in the previous section, there are some restrictions to the use of the robot. It would be interesting, however, to explore the possibilities to use Trilobot for tracking experiments.

- To improve the performance of the robot, encoder disk with a higher resolution can be mounted onto Trilobot. As explained in chapter 4 the hardware on-board Trilobot can keep track of the traveled distance with a much higher resolution using different encoder disks, without effort.

Bibliography

[1] Robert A. Adams. *Calculus: a complete course.* Addison-Wesley, 1995.

[2] Roger Arrick. *Trilobot, Mobile Robot for Research and Education: User Guide.* Arrick robotics, Hurst, Texas, USA, September 1998.

[3] Karl J. Åström and Bjorn Wittenmark. *Computer Controlled Systems: Theory and Design.* Prentice Hall, 1990. ISBN 0-13-172784-2.

[4] Michael S. Branicky, Stephen M. Philips, and Wei Zhang. Stability of networked control systems: Explicit analysis of delay. American Control Conference, pp 2352–2357, June 2000.

[5] Michael S. Branicky and Stephen M. Phillips. Networked control systems repository. Website. `http://home.cwru.edu/ncs/index.htm` Consulted: October 2004.

[6] Michael S. Branicky, Stephen M. Phillips, and Wei Zhang. Scheduling and feedback co-design for networked control systems. 41st IEEE Conference on Decision and Control,pp.1211–1217, December 2002.

[7] Mo-Yuen Chow and Yodyium Tipsuwan. Network-based control systems: A tutorial. 27th Annual Conference of the IEEE Industrial Electronics Society, pp.1593–1602, 2001.

[8] Gene F. Franklin, J. David Powell, and Michael L. Workman. *Digital Control of Dynamic Systems.* Addison-Wesley, 1990. ISBN 0-201-11398-2.

[9] Y. Halevi and A. Ray. Integrated communications and control systems part i - analysis. Journal of Dynamic Systems, Measurements and Control, 110 ,pp.367–373, 1988.

[10] Y. Halevi and A. Ray. Integrated communications and control systems part ii - design considerations. Journal of Dynamic Systems, Measurements and Control, 110 ,pp.374–381, 1988.

[11] H.L.Hagenaars. A trilobot interface in matlab/simulink. Technical Report DCT 2004-64, University of Technology Eindhoven, July 2004.

[12] S.H. Hong and W.-H. Kim. Bandwidth allocation scheme in can protocol. IEE Proc. Control Theory Appl. Vol. 147, No. 1, pp.37–44, Januari 2000.

[13] Eliahu I. Jury. *Inners and Stability of Dynamic Systems.* Wiley-Interscience London, 1974. ISBN 0-471-45335-8.

[14] Brian W. Kernighan and Dennis M. Ritchie. *The C programming language*. Prentice Hall, 1988. ISBN 0-13-110370-9.

[15] Feng-Li Lian, James Moyne, and Dawn Tilbury. Time delay modelling and sample time selection for networked control systems. 2001 International Mechanical Engineering Congres and Exposition, November 2001.

[16] Luen-Woei Liou and Asok Ray. A stochastic regulator for integrated communication and control systems: Part i - formulation of control law. Journal of Dynamic Systems, Measurement and Control, Vol. 113 ,pp.604–611, December 1991.

[17] Luen-Woei Liou and Asok Ray. A stochastic regulator for integrated communication and control systems: Part ii - numerical analysis and simulation. Journal of Dynamic Systems, Measurement and Control, Vol. 113 ,pp.612–619, December 1991.

[18] The Mathworks. *SIMULINK Model-Based and System-Based Design: Writing S-functions*. The Mathworks, Inc., 3 Apple Hill Drive Natick, MA 01760-2098, June 2001.

[19] Microsoft. *Microsoft Devellopers Network Library*. `http://msdn.microsoft.com` Consulted: April 2005.

[20] Luis A. Montestruque and Panos J. Antsaklis. On the model-based control of networked systems. *Automatica*, (39):1837 – 1843, May 2003.

[21] Johan Nilsson. *Real-Time Control Systems with Delays*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, January 1998.

[22] Johan Nilsson. Implement a continuously updating, high resolution time provider for windows. *MSDN Magazine*, March 2004.

[23] Raisonance. *Getting started: 80C52 and XA Development Tools*, January 2000.

[24] Jean-Pierre Richard. Time-delay systems: an overview of some recent advances and open problems. Automatica 39 (2003), pp.1667–1694, April 2003.

[25] Craig Steiner. *8051 Tutorial*. `http://www.8052.com/tut8051.phtml` Consulted: May 2004.

[26] The Analytic Sciences Corporation Technical Staff. *Applied Optimal Estimation*. The M.I.T. Press, 1974.

[27] Yodyium Tipsuwan and Mo-Yuen Chow. Control methodologies in networked control systems. Control Engineering Practice 11, pp.1099–1111, February 2003.

[28] Gregory C. Walsh and Hong Ye. Scheduling of networked control systems. IEEE Control Systems Magazine, pp.57–65, February 2001.

[29] Gregory C. Walsh, Hong Ye, and Linda G. Bushnell. Stability analysis of networked control systems. IEEE Transactions on Control Systems Technology, Vol.10, No.3, May 2002.

[30] Winbond Electronics Corp. *W78E52B Data Sheet*, June 11 2004.

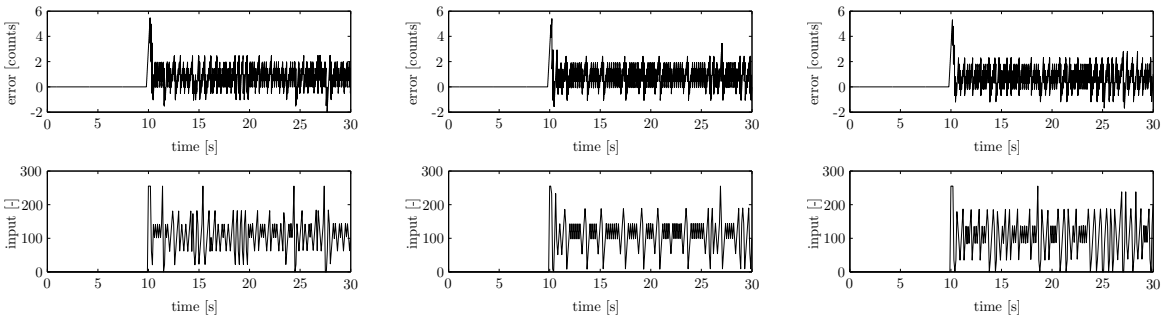
[31] Sencer Yeralan and Ashutosh Ahluwalia. *Programming and Interfacing the 8051 Microcontroller*. Addison-Wesley, 1993. ISBN 0-201-63365-5.

[32] Wei Zhang, Michael S. Branicky, and Stephen M. Phillips. Stability of networked control systems. IEEE Control Systems Magazine, pp.84–99, February 2001.

Appendix A

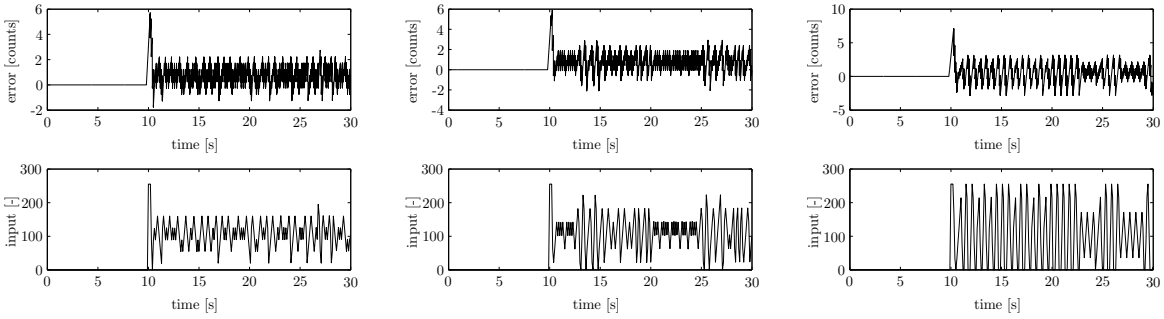
Experimental results

A.1 Results position feedback



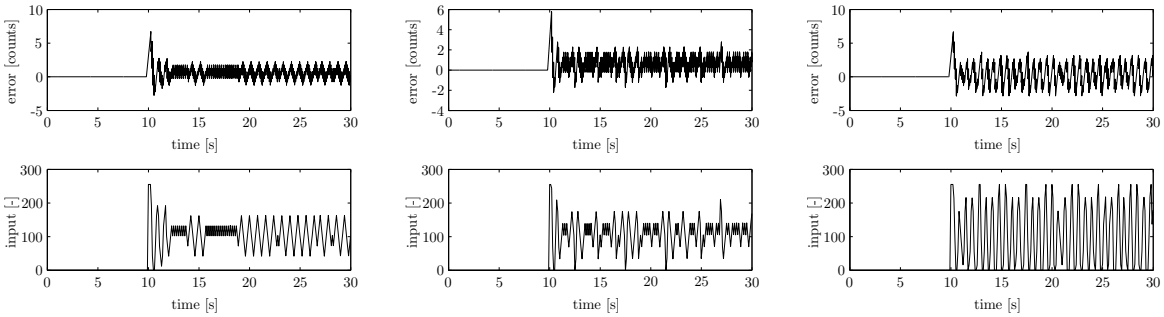
(a) $K_1 = 80$, $\delta_t = 0.005$s (b) $K_1 = 90$, $\delta_t = 0.005$s (c) $K_1 = 100$, $\delta_t = 0.005$s

Figure A.1: Experimental results for $\delta_t = 5\%$.



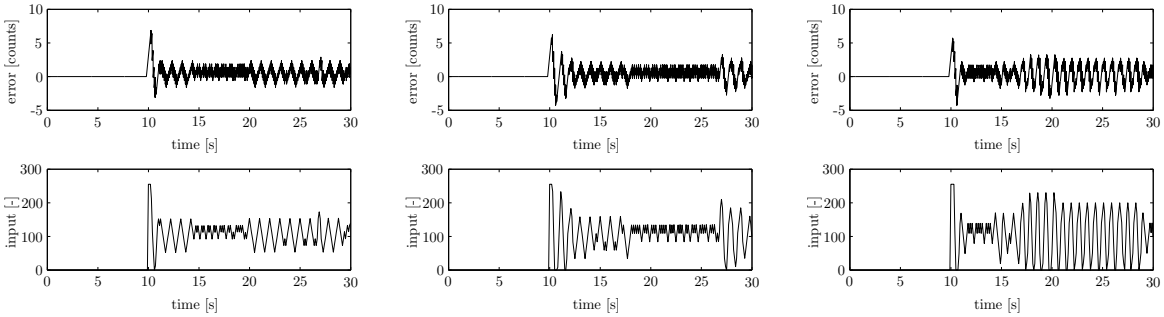
(a) $K_1 = 70$, $\delta_t = 0.01$s (b) $K_1 = 80$, $\delta_t = 0.01$s (c) $K_1 = 90$, $\delta_t = 0.01$s

Figure A.2: Experimental results for $\delta_t = 10\%$.



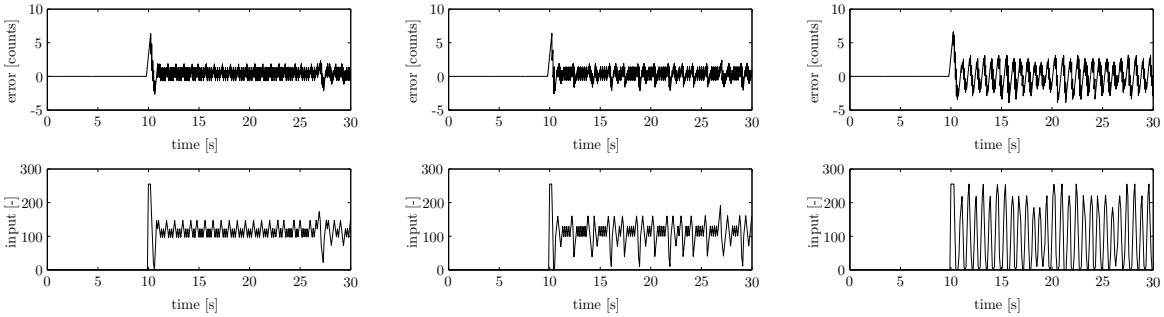
(a) $K_1 = 60$, $\delta_t = 0.02$s (b) $K_1 = 70$, $\delta_t = 0.02$s (c) $K_1 = 80$, $\delta_t = 0.02$s

Figure A.3: Experimental results for $\delta_t = 20\%$.



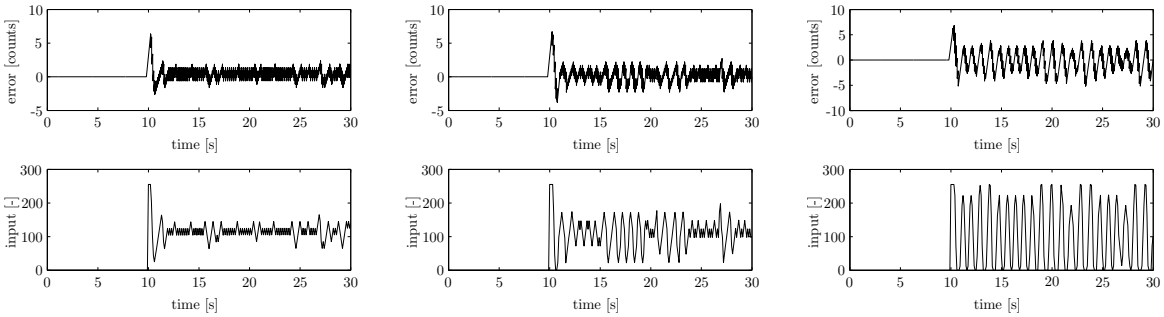
(a) $K_1 = 40$, $\delta_t = 0.03$s (b) $K_1 = 50$, $\delta_t = 0.03$s (c) $K_1 = 60$, $\delta_t = 0.03$s

Figure A.4: Experimental results for $\delta_t = 30\%$.



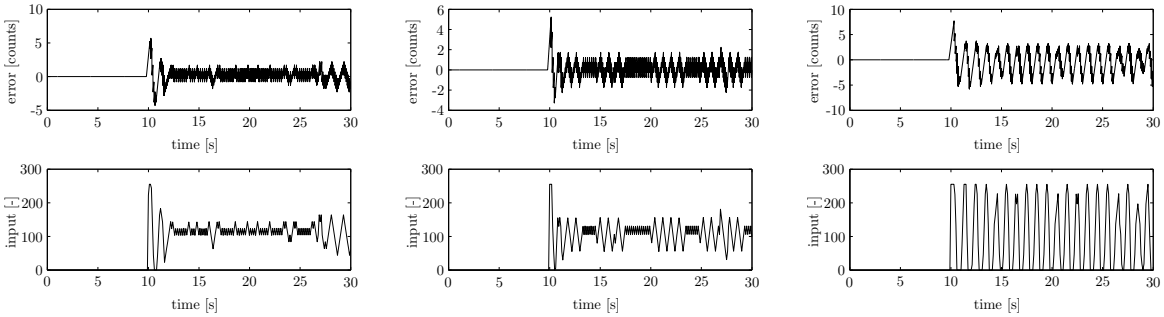
(a) $K_1 = 50$, $\delta_t = 0.04$s (b) $K_1 = 60$, $\delta_t = 0.04$s (c) $K_1 = 70$, $\delta_t = 0.04$s

Figure A.5: Experimental results for $\delta_t = 40\%$.



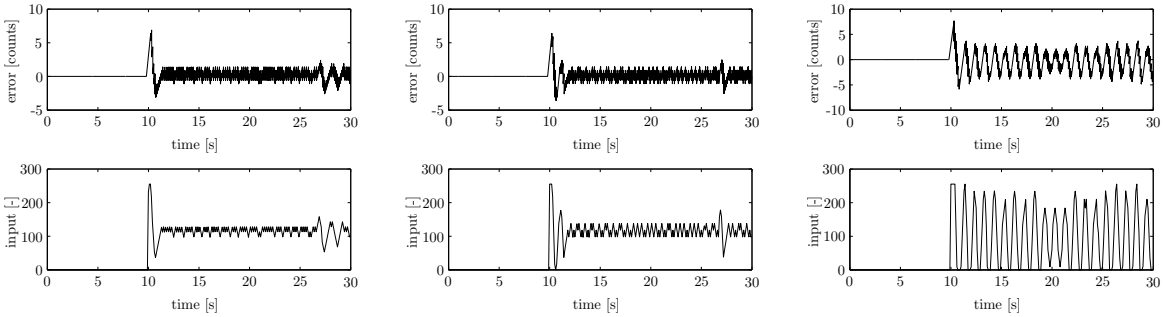
(a) $K_1 = 40$, $\delta_t = 0.05$s (b) $K_1 = 50$, $\delta_t = 0.05$s (c) $K_1 = 60$, $\delta_t = 0.05$s

Figure A.6: Experimental results for $\delta_t = 50\%$.



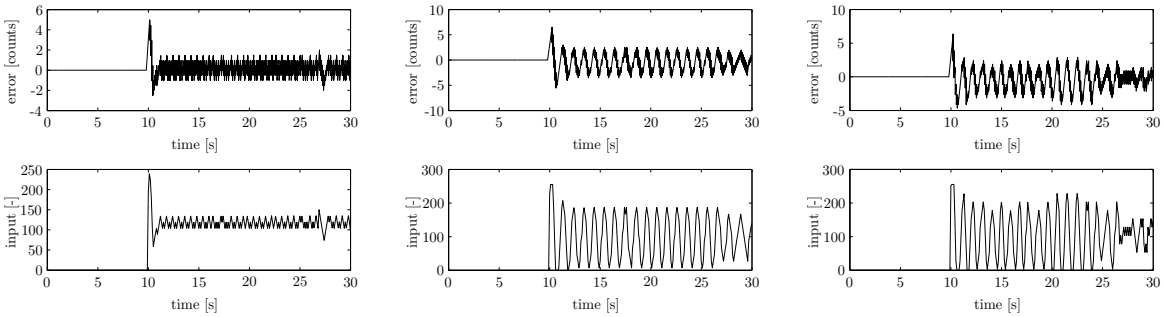
(a) $K_1 = 40$, $\delta_t = 0.06$s (b) $K_1 = 50$, $\delta_t = 0.06$s (c) $K_1 = 60$, $\delta_t = 0.06$s

Figure A.7: Experimental results for $\delta_t = 60\%$.



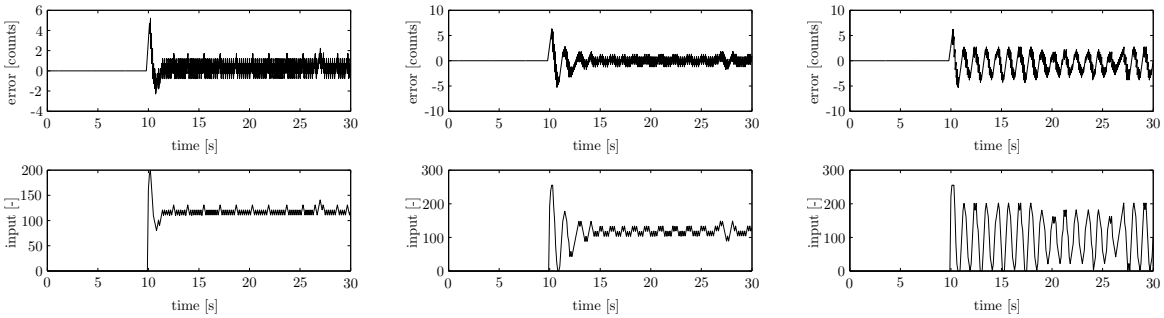
(a) $K_1 = 30$, $\delta_t = 0.07$s (b) $K_1 = 40$, $\delta_t = 0.07$s (c) $K_1 = 50$, $\delta_t = 0.07$s

Figure A.8: Experimental results for $\delta_t = 70\%$.



(a) $K_1 = 30$, $\delta_t = 0.08$s (b) $K_1 = 40$, $\delta_t = 0.08$s (c) $K_1 = 50$, $\delta_t = 0.08$s

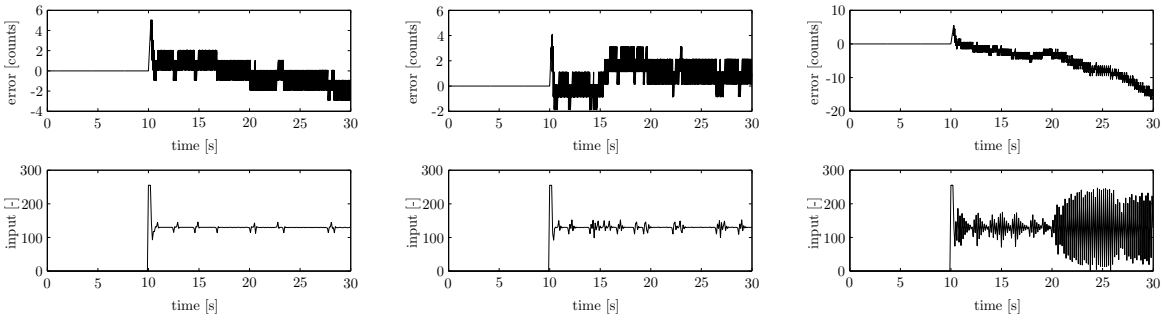
Figure A.9: Experimental results for $\delta_t = 80\%$.



(a) $K_1 = 20$, $\delta_t = 0.09$s (b) $K_1 = 30$, $\delta_t = 0.09$s (c) $K_1 = 40$, $\delta_t = 0.09$s

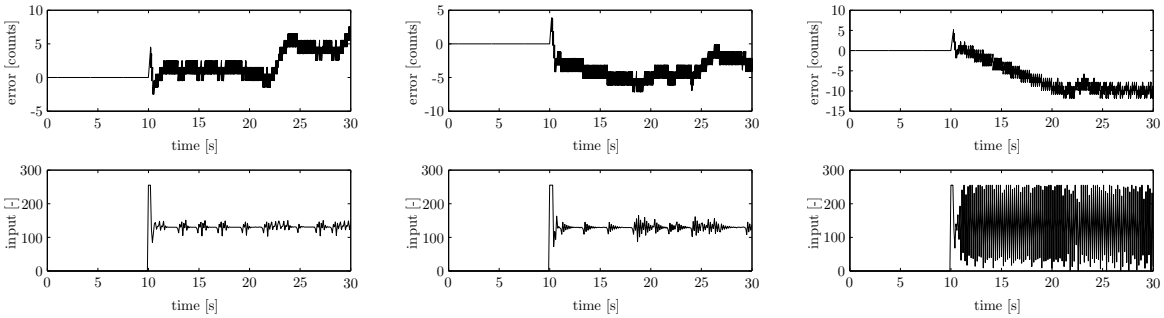
Figure A.10: Experimental results for $\delta_t = 90\%$.

A.2 Results full state feedback



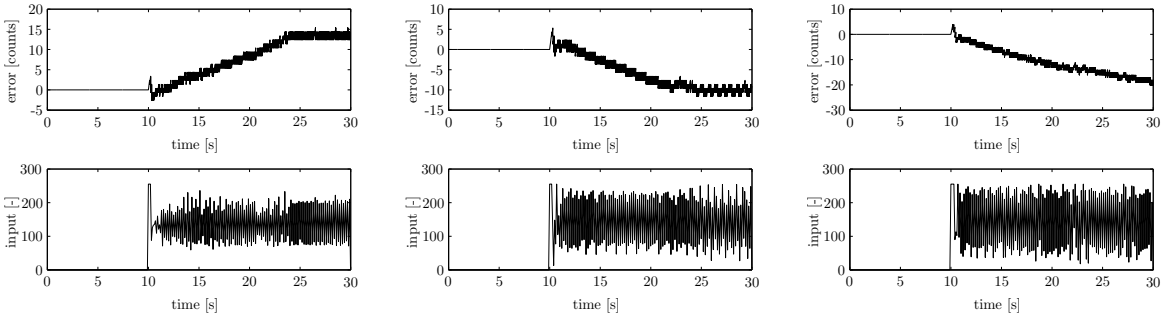
(a) $K_2 = 30$, $\delta_t = 0.005$s (b) $K_2 = 40$, $\delta_t = 0.005$s (c) $K_2 = 50$, $\delta_t = 0.005$s

Figure A.11: Experimental results for $\delta_t = 5\%$.



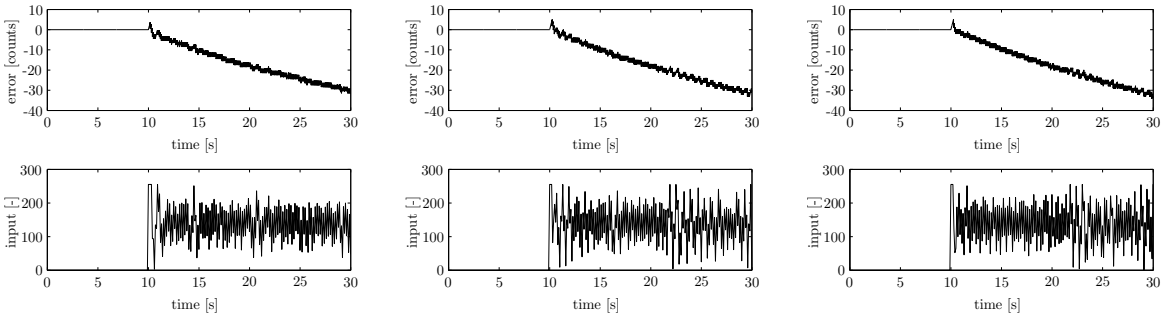
(a) $K_2 = 40$, $\delta_t = 0.01$s (b) $K_2 = 50$, $\delta_t = 0.01$s (c) $K_2 = 60$, $\delta_t = 0.01$s

Figure A.12: Experimental results for $\delta_t = 10\%$.



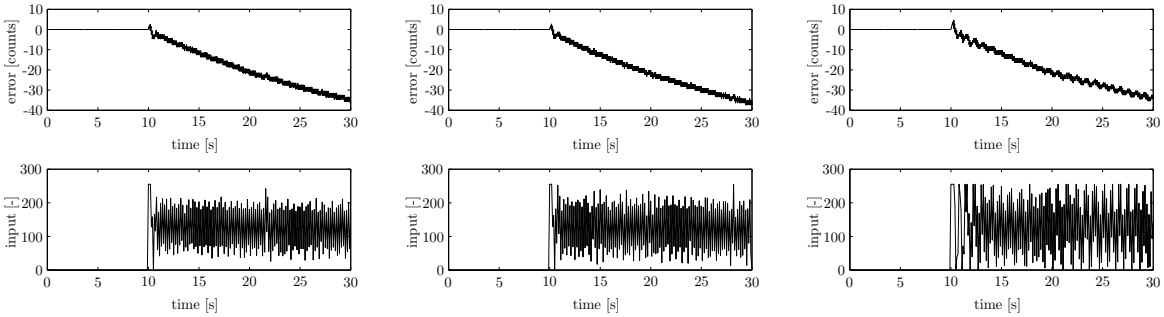
(a) $K_2 = 60$, $\delta_t = 0.015$s (b) $K_2 = 70$, $\delta_t = 0.015$s (c) $K_2 = 80$, $\delta_t = 0.015$s

Figure A.13: Experimental results for $\delta_t = 15\%$.



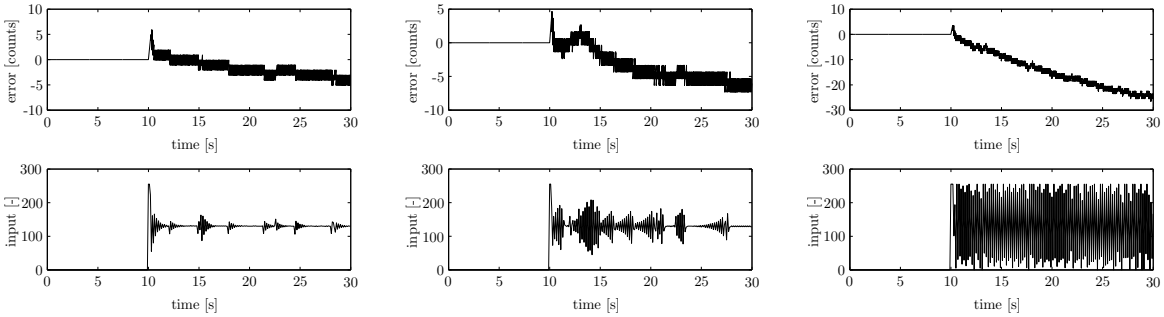
(a) $K_2 = 110$, $\delta_t = 0.02$s (b) $K_2 = 120$, $\delta_t = 0.02$s (c) $K_2 = 130$, $\delta_t = 0.02$s

Figure A.14: Experimental results for $\delta_t = 20\%$.



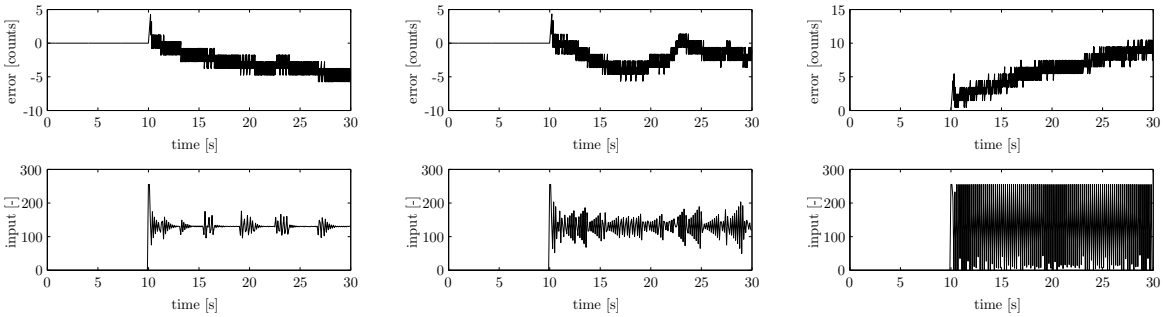
(a) $K_2 = 90$, $\delta_t = 0.03$s (b) $K_2 = 100$, $\delta_t = 0.03$s (c) $K_2 = 110$, $\delta_t = 0.03$s

Figure A.15: Experimental results for $\delta_t = 30\%$.



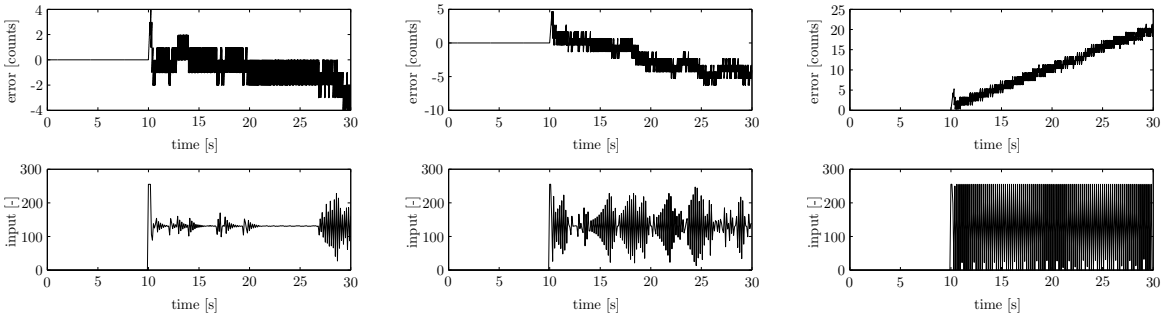
(a) $K_2 = 50$, $\delta_t = 0.04$s (b) $K_2 = 60$, $\delta_t = 0.04$s (c) $K_2 = 70$, $\delta_t = 0.04$s

Figure A.16: Experimental results for $\delta_t = 40\%$.



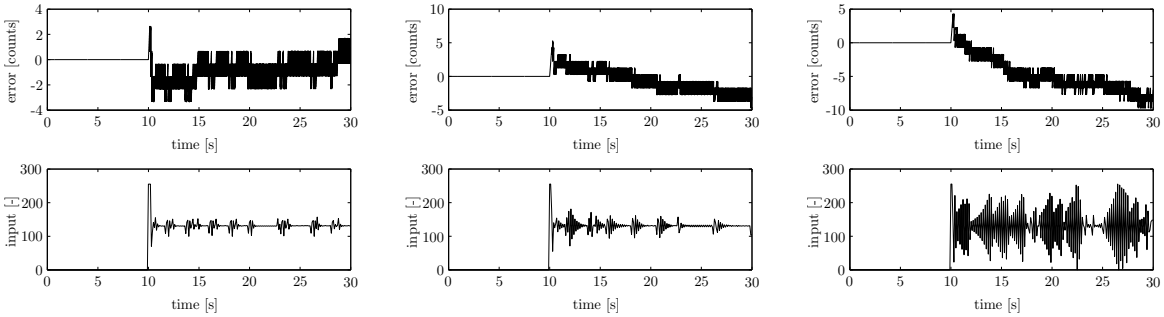
(a) $K_2 = 50$, $\delta_t = 0.05$s (b) $K_2 = 60$, $\delta_t = 0.05$s (c) $K_2 = 70$, $\delta_t = 0.05$s

Figure A.17: Experimental results for $\delta_t = 50\%$.



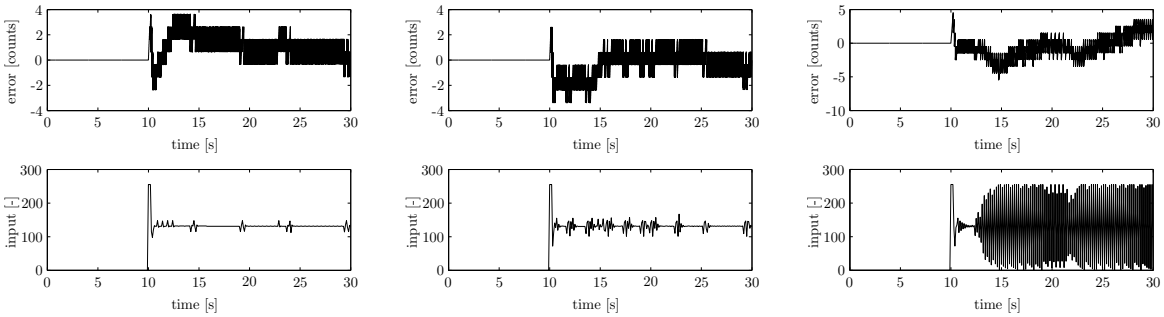
(a) $K_2 = 50$, $\delta_t = 0.06$s (b) $K_2 = 60$, $\delta_t = 0.06$s (c) $K_2 = 70$, $\delta_t = 0.06$s

Figure A.18: Experimental results for $\delta_t = 60\%$.



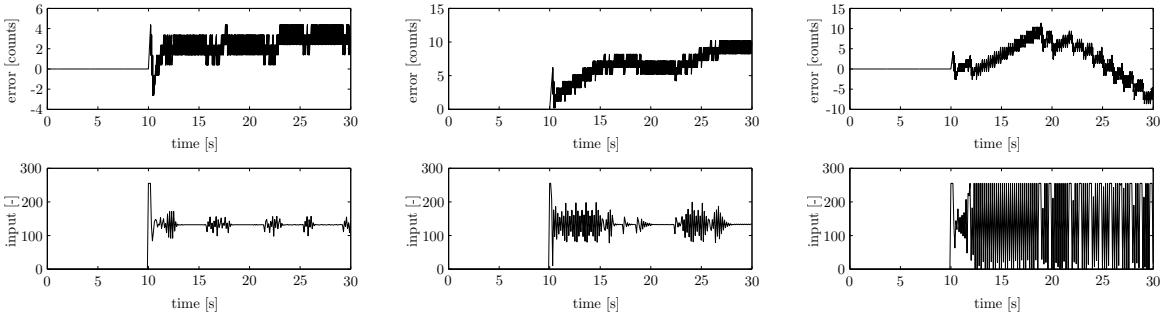
(a) $K_2 = 40$, $\delta_t = 0.07$s (b) $K_2 = 50$, $\delta_t = 0.07$s (c) $K_2 = 60$, $\delta_t = 0.07$s

Figure A.19: Experimental results for $\delta_t = 70\%$.



(a) $K_2 = 30$, $\delta_t = 0.08$s (b) $K_2 = 40$, $\delta_t = 0.08$s (c) $K_2 = 50$, $\delta_t = 0.08$s

Figure A.20: Experimental results for $\delta_t = 80\%$.



(a) $K_2 = 40$, $\delta_t = 0.09$s (b) $K_2 = 50$, $\delta_t = 0.09$s (c) $K_2 = 60$, $\delta_t = 0.09$s

Figure A.21: Experimental results for $\delta_t = 90\%$.

Appendix B

Proofs

$$\lim_{h\to 0}\left[1+\frac{\log\left(\frac{3+e^h}{2e^h+1+e^{2h}}\right)}{h}\right]=\frac{1}{4}. \tag{B.1}$$

Proof:

$$\lim_{h\to 0}\left[1+\frac{\log\left(\frac{3+e^h}{2e^h+1+e^{2h}}\right)}{h}\right]=1+\lim_{h\to 0}\left[\frac{\log\left(\frac{3+e^h}{2e^h+1+e^{2h}}\right)}{h}\right] \tag{B.2}$$

Applying l'Hôpital's rule [1]:

$$\lim_{h\to 0}\left[\frac{\log\left(\frac{3+e^h}{2e^h+1+e^{2h}}\right)}{h}\right]=\frac{d}{dh}\left[\log\left(\frac{3+e^h}{2e^h+1+e^{2h}}\right)\right]\Big|_{h=0} \tag{B.3}$$

Applying the chain rule:

$$\begin{aligned}\frac{d}{dh}\left[\log\left(\frac{3+e^h}{2e^h+1+e^{2h}}\right)\right]\Big|_{h=0} &= \frac{2e^h+1+e^{2h}}{3+e^h}\frac{d}{dh}\left[\frac{3+e^h}{2e^h+1+e^{2h}}\right]\Big|_{h=0} \\ &= \frac{d}{dh}\left[\frac{3+e^h}{2e^h+1+e^{2h}}\right]\Big|_{h=0}\end{aligned} \tag{B.4}$$

Applying the quotient rule:

$$\frac{d}{dh}\left[\frac{3+e^h}{2e^h+1+e^{2h}}\right]\Big|_{h=0}=\frac{(2e^h+1+e^{2h})e^h-(3+e^h)(2e^h+2e^h)}{(2e^h+1+e^{2h})^2}\Big|_{h=0}=-\frac{3}{4} \tag{B.5}$$

Substituting this result back in (B.2) gives:

$$\lim_{h\to 0}\left[1+\frac{\log\frac{3+e^h}{2e^h+1+e^{2h}}}{h}\right]=\frac{1}{4}\qquad\square. \tag{B.6}$$

Appendix C

The Jury stability test

C.1 Jury's test

A discrete-time system is stable if al the roots of its characteristic polynomial lie inside the unit-circle. To test this condition without calculating the roots, Jury's test can be used. In this appendix the use of the Jury test will be illustrated. The mathematical background of the Jury test can be found in [13].
Let $a(z) = a_0 z^n + a_1 z^{n-1} + \cdots + a_n$ be the characteristic polynomial of some discrete-time system. To apply Jury's test, first make sure a_0 is positive by multiplying $a(z)$ by -1 if necessary. Then form rows of the coefficients as follows:

$$\begin{array}{cccc} a_0 & a_1 & \cdots & a_n \\ a_n & a_{n-1} & \cdots & a_0 \\ b_0 & b_1 & \cdots & \cdot \\ b_{n-1} & b_{n-2} & \cdots & \cdot \end{array}$$

Each even row is equal to its preceding odd row in reverse order. The entries in the third row are formed from the second-order determinants using the first column of the first two rows with each of the other columns from these rows, starting from the right and dividing by a_0. The third row can be computed using the following formulas:

$$\begin{aligned} b_0 &= a_0 - \frac{a_n}{a_0} a_n, \\ b_1 &= a_1 - \frac{a_n}{a_0} a_{n-1}, \\ b_k &= a_k - \frac{a_n}{a_0} a_{n-k}. \end{aligned}$$

The fourth row is equal to the third row in reverse order.
The computation of the elements of the fifth row is straightforward. They can be computed as follows:

$$c_k = b_k - \frac{b_{n-1}}{b_0} b_{n-1-k}.$$

The resulting array is called the Jury array. The original polynomial is stable if all the terms in the first column of the odd rows of the Jury array are positive, i.e. $a_0 > 0, b_0 > 0, c_0 > 0, \ldots$

[8]
Because MATLAB does not provide an implementation of the Jury test, a function is written to compute the Jury array either numerically or using a symbolic expression of the characteristic polynomial. The code is listed in the following section.

C.2 jury.m

```
function RA=jury(poli)

if(nargin > 2),
    fprintf('\nError: Too many input arguments given. ...
             ... \nType "help jury" for more information.');
    return
end

dim=size(poli);

coeff=dim(2);
RA=sym(zeros(2*coeff-1,coeff));

for i=1:coeff,
    RA(1,i)=poli(i);
    RA(2,i)=poli(coeff+1-i);
end

for i=3:2*coeff-1,

    if rem(i,2)
        for j=1:(coeff-ceil((i-2)/2))
            RA(i,j)=RA(i-2,j)-(RA(i-2,coeff-floor((i-2)/2))/ ...
            ... RA(i-2,1))*RA(i-2,coeff-floor((i-2)/2)-j+1);
        end
    end

    if ~rem(i,2)
        for j=1:(coeff-ceil((i-2)/2))
            RA(i,j)= RA(i-1,coeff-(i/2-1));
        end
    end
end

if isempty(findsym(RA))
   RA = double(RA);
end
```

Appendix D

Windows2000 DLL's

In this appendix a code-listing is given of the two DLL's in which the routines to communicate with Trilobot are defined. These DLL's are written for and tested under the Windows2000 operating system [19, 14, 22].

D.1 Single wheel routines [triloSingle.dll]

```
#define STRICT
#define MAX 1024;

#include <assert.h>
#include <memory.h>
#include <process.h>
#include <tchar.h>
#include <dos.h>
#include <conio.h>
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include <stdlib.h>
#include <time.h>

static double LEFT, LEFTDOT;
static int   TERM, COMWRITE, READLEFT, LOG, DELAY;
char *port = "com1";
int Left[10], iterator;
double freq, time1, time2, time3, time4;
long start;
signed int WriteLeft[10];
unsigned long n_in, n_out;
HANDLE hPort;
HANDLE hThread;
DCB dcb;
FILE *fidin, *fidout;

struct reference_point
{
  FILETIME file_time;
  LARGE_INTEGER counter;
};

LARGE_INTEGER   frequency;
reference_point ref_point;
```

```
FILETIME        file_time1;
SYSTEMTIME      system_time1;
FILETIME        file_time2;
SYSTEMTIME      system_time2;


void Synchronize(reference_point& ref_point)
{
  FILETIME      ft0 = { 0, 0 },
                ft1 = { 0, 0 };
  LARGE_INTEGER li;

  ::GetSystemTimeAsFileTime(&ft0);
  do
  {
    ::GetSystemTimeAsFileTime(&ft1);
    ::QueryPerformanceCounter(&li);
  }
  while ((ft0.dwHighDateTime == ft1.dwHighDateTime) &&
         (ft0.dwLowDateTime == ft1.dwLowDateTime));

  ref_point.file_time = ft1;
  ref_point.counter = li;
}


void get_time(LARGE_INTEGER frequency, const reference_point& reference, FILETIME& current_time)
{
  LARGE_INTEGER li;

  ::QueryPerformanceCounter(&li);

  LARGE_INTEGER ticks_elapsed;

  ticks_elapsed.QuadPart = li.QuadPart -
      reference.counter.QuadPart;

  ULARGE_INTEGER filetime_ticks,
                 filetime_ref_as_ul;

  filetime_ticks.QuadPart =
      (ULONGLONG)((((double)ticks_elapsed.QuadPart/(double)
      frequency.QuadPart)*10000000.0)+0.5);
  filetime_ref_as_ul.HighPart = reference.file_time.dwHighDateTime;
  filetime_ref_as_ul.LowPart = reference.file_time.dwLowDateTime;
  filetime_ref_as_ul.QuadPart += filetime_ticks.QuadPart;


  current_time.dwHighDateTime = filetime_ref_as_ul.HighPart;
  current_time.dwLowDateTime = filetime_ref_as_ul.LowPart;
}

void sec_init(void)
{
    LARGE_INTEGER lFreq, lCnt;
    QueryPerformanceFrequency(&lFreq);
    freq = (double)lFreq.LowPart;
    QueryPerformanceCounter(&lCnt);
    start = lCnt.LowPart;
}

double sec(void)
{
    LARGE_INTEGER lCnt;
    long tcnt;
    QueryPerformanceCounter(&lCnt);
```

```
    tcnt = lCnt.LowPart - start;
    return ((double)tcnt) / freq;
}

void delay(int delay_time)
{
    double begin_time, end_time;
    begin_time = 1000*sec();
    end_time   = begin_time + (double)delay_time;
    while(begin_time < end_time)
    {
        begin_time = 1000*sec();
    }
    return;
}

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}

static DWORD single_TRILOThreadID;
DWORD single_TRILOThread(LPVOID);


extern "C" _declspec(dllexport) void single_TRILOStart(void)
{
    hThread = CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) single_TRILOThread,
              (LPVOID) NULL,0,&single_TRILOThreadID);

}

extern "C" _declspec(dllexport) void single_TRILOOpen(int Baud, int PortID, int Log, int Delay)
{
    LOG = Log;
    DELAY = (Delay-1)*10;
    iterator = 0;

    if(PortID == 1){port = "com1";}
    if(PortID == 2){port = "com2";}
    else{port = "com1";}

  hPort = CreateFile(port, GENERIC_READ|GENERIC_WRITE,
                           0,
                           NULL,
                           TRUNCATE_EXISTING,
                           FILE_ATTRIBUTE_NORMAL,
                           NULL);

   if (hPort == INVALID_HANDLE_VALUE)
   {
       printf ("CreateFile failed with error %d.\n", GetLastError());
   }
```

```
   if (!GetCommState(hPort, &dcb))
   {
      printf ("GetCommState failed with error %d.\n", GetLastError());
   }

   dcb.BaudRate = CBR_9600;

   if(Baud == 1 ) {dcb.BaudRate = CBR_1200;}
   if(Baud == 2 ) {dcb.BaudRate = CBR_4800;}
   if(Baud == 3 ) {dcb.BaudRate = CBR_9600;}
   if(Baud == 4 ) {dcb.BaudRate = CBR_19200;}

   dcb.ByteSize = 8;
   dcb.Parity = NOPARITY;
   dcb.StopBits = ONESTOPBIT;

   COMMTIMEOUTS timeouts;
   timeouts.ReadIntervalTimeout = 0;
   timeouts.ReadTotalTimeoutMultiplier = 0;
   timeouts.ReadTotalTimeoutConstant = 0;
   timeouts.WriteTotalTimeoutMultiplier = 0;
   timeouts.WriteTotalTimeoutConstant = 0;

   if (!SetCommState(hPort, &dcb))
   {
      printf ("SetCommState failed with error %d.\n", GetLastError());
   }
   else
   {
        printf("Serial port %s (re)configured: BaudRate %d, ByteSize %d, Parity %d.\n",
                port,dcb.BaudRate,dcb.ByteSize,dcb.Parity);
   }


   if (!SetCommTimeouts(hPort, &timeouts))
   {
      printf ("SetCommTimeouts failed with error %d.\n", GetLastError());
   }

   if(LOG)
   {
    fidin = fopen("D:\\data\\input.txt","w");
    fidout = fopen("D:\\data\\output.txt","w");
    if(!fidin)
    {
         printf("Could not open input.txt for data logging \n");
    }
    if(!fidout)
    {
         printf("Could not open output.txt for data logging\n");
    }
    if(!fidsample)
    {
         printf("Could not open sample.txt for data logging\n");
    }
    else {printf("Files openend for data logging.\n");}
   }

   WriteLeft[0]  = 0;
   LEFT = 0.0;

   ::QueryPerformanceFrequency(&frequency);
   Synchronize(ref_point);
   sec_init();

}
```

```
DWORD single_TRILOThread(LPVOID param)
{

   TERM = 0;

   double prevleft, overflowLeft, prevleftdot, leftestim, prevleftestim;
   prevleft = overflowLeft = prevleftdot =  leftestim = prevleftestim = 0.0;

   Left[0]  = 0;

    while (TERM==0)
    {

        n_in=0;
        n_out=0;

        READLEFT   = ReadFile(hPort, Left , 1, &n_out, NULL);
        if(prevleft  > (double) Left[0] ) overflowLeft = overflowLeft + 1.0;

        LEFT  = overflowLeft*255  + (double) Left[0];

        prevleft  = (double) Left[0];

        leftestim = prevleftestim + 0.08944*prevleftdot + 0.001612*WriteLeft[0] +1*(LEFT - prevleftestim);
        LEFTDOT = 0.7966*prevleftdot+0.03106*WriteLeft[0] + 0.5*(LEFT-prevleftestim);

        prevleftdot = LEFTDOT;
        prevleftestim = leftestim;

        get_time(frequency, ref_point, file_time1);
        ::FileTimeToSystemTime(&file_time1, &system_time1);

        time2 = time1;
        time1 = system_time1.wMinute*60*1000 + system_time1.wSecond*1000 + system_time1.wMilliseconds;

        if(LOG){fprintf(fidout,"%f \t %f \t %f\n",LEFT,time1,time1-time2);}

        delay(DELAY);

        COMWRITE = WriteFile(hPort, WriteLeft, 1, &n_out, NULL);

        get_time(frequency, ref_point, file_time2);
        ::FileTimeToSystemTime(&file_time2, &system_time2);

        time4 = time3;
        time3 = system_time2.wMinute*60*1000 + system_time2.wSecond*1000 + system_time2.wMilliseconds;

        if(LOG)
        {
            fprintf(fidin,"%i \t %f \t %f\n",WriteLeft[0],time3,time3-time4);
        }

    }

    if (LOG)
    {
        fclose(fidin);
        fclose(fidout);
    }

    if (!CloseHandle(hPort))
    {
       printf ("CloseHandle failed with error %d.\n", GetLastError());
    }
    else
```

```
    {
       printf ("The serial communication has been stopped");
    }
    Sleep(500);
    return 0;
}


extern "C" _declspec(dllexport) void single_TRILOLeft(double* Get_Left)
{
    Get_Left[0]=LEFT;
}

extern "C" _declspec(dllexport) void single_TRILOLeftDot(double* Get_Left_Dot)
{
    Get_Left_Dot[0]=LEFTDOT;
}

extern "C" _declspec(dllexport) void single_TRILOSend(int Lft)
{
    WriteLeft[0]  = Lft;
}

extern "C" _declspec(dllexport) void single_TRILOTerm(void)
{
    WriteLeft[0]  = 0;
    printf("\nStopping the drivemotor...\n");
    COMWRITE = WriteFile(hPort, WriteLeft, 1, &n_out, NULL);
    Sleep(1000);
    printf("\nStopping the read operations...\n");
    TERM = 1;
}
```

D.2 Double wheel routines [trilo.dll]

```
#define STRICT
#define MAX 1024;

#include <assert.h>
#include <memory.h>
#include <process.h>
#include <tchar.h>
#include <dos.h>
#include <conio.h>
#include <windows.h>
#include <stdio.h>
#include <string.h>
#include <iostream.h>
#include <stdlib.h>
#include <time.h>

static double LEFT, RIGHT;
static int   TERM, COMWRITE, READLEFT, READRIGHT, LOG, DELAY;
char *port = "com1";
int Left[10];
int Right[10];
double freq;
long start;
signed int WriteLeft[10];
signed int WriteRight[10];
unsigned long n_in, n_out;
HANDLE hPort;
DCB dcb;
FILE *fidin, *fidout;
```

```
void sec_init(void)
{
    LARGE_INTEGER lFreq, lCnt;
    QueryPerformanceFrequency(&lFreq);
    freq = (double)lFreq.LowPart;
    QueryPerformanceCounter(&lCnt);
    start = lCnt.LowPart;
}

double sec(void)
{
    LARGE_INTEGER lCnt;
    long tcnt;
    QueryPerformanceCounter(&lCnt);
    tcnt = lCnt.LowPart - start;
    return ((double)tcnt) / freq;
}

void delay(int delay_time)
{
    double begin_time, end_time;
    begin_time = 1000*sec();
    end_time   = begin_time + (double)delay_time;
    while(begin_time < end_time)
    {
        begin_time = 1000*sec();
    }
    return;
}

BOOL APIENTRY DllMain( HANDLE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved
                     )
{
    switch (ul_reason_for_call)
    {
        case DLL_PROCESS_ATTACH:
        case DLL_THREAD_ATTACH:
        case DLL_THREAD_DETACH:
        case DLL_PROCESS_DETACH:
        break;
    }
    return TRUE;
}

static DWORD TRILOThreadID;
DWORD TRILOThread(LPVOID);

extern "C" _declspec(dllexport) void TRILOStart(void)
{
    CreateThread(NULL,0,(LPTHREAD_START_ROUTINE) TRILOThread,
             (LPVOID) NULL,0,&TRILOThreadID);
}

extern "C" _declspec(dllexport) void TRILOOpen(int Baud, int PortID, int Log, int Delay)
{
    sec_init();

    LOG = Log;
    DELAY = Delay;

    if(PortID == 1){port = "com1";}
    if(PortID == 2){port = "com2";}
    else{port = "com1";}
```

```
hPort = CreateFile(port, GENERIC_READ|GENERIC_WRITE,
                         0,
                         NULL,
                         TRUNCATE_EXISTING,
                         FILE_ATTRIBUTE_NORMAL,
                         NULL);

 if (hPort == INVALID_HANDLE_VALUE)
 {
     printf ("CreateFile failed with error %d.\n", GetLastError());
 }


 if (!GetCommState(hPort, &dcb))
 {
    printf ("GetCommState failed with error %d.\n", GetLastError());
 }

 dcb.BaudRate = CBR_9600;

 if(Baud == 1 ) {dcb.BaudRate = CBR_1200;}
 if(Baud == 2 ) {dcb.BaudRate = CBR_4800;}
 if(Baud == 3 ) {dcb.BaudRate = CBR_9600;}
 if(Baud == 4 ) {dcb.BaudRate = CBR_19200;}

 dcb.ByteSize = 8;
 dcb.Parity = NOPARITY;
 dcb.StopBits = ONESTOPBIT;

 COMMTIMEOUTS timeouts;
 timeouts.ReadIntervalTimeout = 0;
 timeouts.ReadTotalTimeoutMultiplier = 0;
 timeouts.ReadTotalTimeoutConstant = 0;
 timeouts.WriteTotalTimeoutMultiplier = 0;
 timeouts.WriteTotalTimeoutConstant = 0;

 if (!SetCommState(hPort, &dcb))
 {
    printf ("SetCommState failed with error %d.\n", GetLastError());
 }
 else
 {
      printf("Serial port %s (re)configured: BaudRate %d, ByteSize %d, Parity %d.\n",
              port,dcb.BaudRate,dcb.ByteSize,dcb.Parity);
 }


 if (!SetCommTimeouts(hPort, &timeouts))
 {
    printf ("SetCommTimeouts failed with error %d.\n", GetLastError());
 }

 if(LOG)
 {
  fidin = fopen("input.txt","w");
  fidout = fopen("output.txt","w");
  if(!fidin)
  {
       printf("Could not open input.txt for data logging \n");
  }
  if(!fidout)
  {
       printf("Could not open output.txt for data logging\n");
  }
  else {printf("Files openend for data logging.\n");}
```

```
   }

   WriteLeft[0]  = 0;
   WriteRight[0] = -1;
   LEFT = 0.0;
   RIGHT = 0.0;

}

DWORD TRILOThread(LPVOID param)
{

   TERM = 0;

   double prevleft, prevright, overflowLeft, overflowRight;
   prevleft = prevright = overflowLeft = overflowRight = 0.0;

   Left[0]  = 0;
   Right[0] = 0;

    while (TERM==0)
    {
        double time1, time2;
        time1 = sec();

        n_in=0;
        n_out=0;

        READRIGHT  = ReadFile(hPort, Right, 1, &n_out, NULL);
        READLEFT   = ReadFile(hPort, Left , 1, &n_out, NULL);;

        if(prevleft  > (double) Left[0] ) overflowLeft = overflowLeft + 1.0;
        if(prevright > (double) Right[0]) overflowRight= overflowRight+ 1.0;

        LEFT  = overflowLeft*255  + (double) Left[0];
        RIGHT = overflowRight*255 + (double) Right[0];

        prevleft  = (double) Left[0];
        prevright = (double) Right[0];

        time2 = sec();

        if(LOG){fprintf(fidout,"%f \t %f \t %f\n",LEFT,RIGHT,1000*(time2-time1));}

    }

    if (LOG)
    {
        fclose(fidin);
        fclose(fidout);
    }

    if (!CloseHandle(hPort))
    {
       printf ("CloseHandle failed with error %d.\n", GetLastError());
    }
    else
    {
       printf ("The serial communication has been stopped");
    }
    Sleep(500);
    return 0;
}
extern "C" _declspec(dllexport) void TRILOLeft(double* Get_Left)
{
    Get_Left[0]=LEFT;
```

```
}
extern "C" _declspec(dllexport) void TRILORight(double* Get_Right)
{
    Get_Right[0]=RIGHT;
}
extern "C" _declspec(dllexport) void TRILOSend(int Lft, int Rgt)
{
    double time1, time2;
    time1 = sec();
    delay(DELAY);
    WriteLeft[0]  = Lft;
    WriteRight[0] = Rgt;
    COMWRITE = WriteFile(hPort, WriteLeft, 1, &n_out, NULL);
    COMWRITE = WriteFile(hPort, WriteRight, 1, &n_out, NULL);
    time2 = sec();
    if(LOG)
    {
        fprintf(fidin,"%i \t %i \t %f\n",WriteLeft[0],WriteRight[0],1000*(time2-time1));
    }
}
extern "C" _declspec(dllexport) void TRILOTerm(void)
{
    WriteLeft[0]  = 0;
    WriteRight[0] = -1;
    printf("\nStopping the drivemotors...\n");
    COMWRITE = WriteFile(hPort, WriteLeft, 1, &n_out, NULL);
    Sleep(1000);
    COMWRITE = WriteFile(hPort, WriteRight, 1, &n_out, NULL);
    Sleep(1000);
    printf("\nStopping the read operations...\n");
    TERM = 1;
}
```

Appendix E

Trilobot C-MEX S-Functions

In this appendix a code listing is given of the four C-MEX S-functions [18] written for the Simulink interface for the single-wheel setup as well as the two-wheel setup. In these S-functions the routines defined in `triloSingle.dll` and `trilo.dll` (see appendix D) are invoked.

E.1 s_single_in.c

```
#define S_FUNCTION_NAME s_single_in
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"

#define TS2    ssGetSFcnParam(S, 0)

#define NSTATES     0
#define NINPUTS     1
#define NOUTPUTS    0
#define NPARAMS     1

#include <math.h>

extern void single_TRILOSend(uint8_T);

static void mdlInitializeSizes(SimStruct *S)
{

    ssSetNumSFcnParams(S, NPARAMS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))return;

    ssSetNumContStates(S,0);
    ssSetNumDiscStates(S,0);

    if (!ssSetNumInputPorts(S,1)) return;
    ssSetInputPortWidth(S,0,1);
    ssSetInputPortDirectFeedThrough(S,0,0);

    if (!ssSetNumOutputPorts(S,0)) return;

    ssSetInputPortDataType(S, 0, SS_UINT8);

    ssSetNumSampleTimes(S,1);
    ssSetNumRWork(S,0);
```

```
    ssSetNumIWork(S,0);
    ssSetNumPWork(S,0);
    ssSetNumModes(S,0);
    ssSetNumNonsampledZCs(S,0);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S,0, (real_T) (*(mxGetPr(TS2))));
    ssSetOffsetTime(S,0,0.0);
}

static void mdlOutputs(SimStruct *S, int_T tid)
{
      InputPtrsType      uLeft     = ssGetInputPortSignalPtrs(S,0);
      InputInt8PtrsType  uLeftInt8 = (InputInt8PtrsType)uLeft;

#ifndef MATLAB_MEX_FILE
        single_TRILOSend(*uLeftInt8[0]);
#endif
}

static void mdlTerminate(SimStruct *S)
{
   return;
}

#ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

E.2 s_single_out.c

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME  s_single_out

#include "simstruc.h"
#include <string.h>
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#define BAUD  ssGetSFcnParam(S, 0)
#define TS    ssGetSFcnParam(S, 1)
#define LOG   ssGetSFcnParam(S, 2)
#define COM   ssGetSFcnParam(S, 3)
#define DELAY ssGetSFcnParam(S, 4)

#define NSTATES     0
#define NINPUTS     0
#define NOUTPUTS    2
#define NPARAMS     5

extern void single_TRILOStart(void);
extern void single_TRILOOpen(int_T, int_T, int_T, int_T);
extern void single_TRILOLeft(double*);
extern void single_TRILOLeftDot(double*);
extern void single_TRILORight(double*);
extern void single_TRILOTerm(void);

static void mdlInitializeSizes(SimStruct *S)
{
```

```
#ifndef  MATLAB_MEX_FILE

    int_T Baud, Log, Com, Delay;
    real_T Ts;
    int_T Baud_Range[4]={1200,4800,9600,19200};

    Baud = (int_T) (*(mxGetPr(BAUD)));
    Log  = (int_T) (*(mxGetPr(LOG)));
    Com  = (int_T) (*(mxGetPr(COM)));
    Ts   = (real_T) (*(mxGetPr(TS)));
    Delay= (int_T) (*(mxGetPr(DELAY)));

printf("\n\n################################################################################\n");
    printf("Trilobot Interface Configuration Settings\t:\n\n");
    printf("Selected Communication port\t:\tCOM%i\n",Com);
    printf("Selected Console BaudRate\t:\t%i Baud\n",Baud_Range[Baud-1]);
    printf("Selected block sample-time\t:\t%.4f  [%.0f Hz]\n",Ts,1/Ts);
    printf("Selected additional delay\t:\t%i percent of sample time\n",(Delay-1)*10);

    if(Log == 1)
    {
        printf("Real-time data logging\t\t:\tenabled\n");
    }
    else{printf("Real-time data logging\t\t:\tdisabled\n");}

    printf("Make sure Trilobot runs at 10Hz\n");
    printf("\n\n################################################################################\n");
    Sleep(1000);

    printf("\nInitializing communication with Trilobot\n");

    single_TRILOOpen(Baud,Com,Log,Delay);

    printf("Ready to experiment !!!\n");

    single_TRILOStart();

#endif

    ssSetNumSFcnParams(S, NPARAMS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))return;

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if (!ssSetNumInputPorts(S, 0)) return;

    if (!ssSetNumOutputPorts(S, 2)) return;
    ssSetOutputPortWidth(S, 0, 1);
    ssSetOutputPortWidth(S, 1, 1);

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{

    ssSetSampleTime(S, 0, (real_T) (*(mxGetPr(TS))));
```

```
    ssSetOffsetTime(S, 0, 0.0);
}

static void mdlOutputs(SimStruct *S, int_T tid)
{
    double left, leftdot;
    real_T *leftlong = ssGetOutputPortRealSignal(S,0);
    real_T *leftdotlong = ssGetOutputPortRealSignal(S,1);

#ifndef  MATLAB_MEX_FILE
    single_TRILOLeft(&left);
    leftlong[0] = left;
    single_TRILOLeftDot(&leftdot);
    leftdotlong[0] = leftdot;
#endif
}

static void mdlTerminate(SimStruct *S)
{
    #ifndef  MATLAB_MEX_FILE
        single_TRILOTerm();
        sleep(1000);
        printf("\nTRILO system disconnected.\n");
    #endif
}


#ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

E.3 s_trilo_in.c

```
#define S_FUNCTION_NAME s_trilo_in
#define S_FUNCTION_LEVEL 2

#include "simstruc.h"

#define TS2    ssGetSFcnParam(S, 0)

#define NSTATES     0
#define NINPUTS     1
#define NOUTPUTS    0
#define NPARAMS     1

#include <math.h>

extern void TRILOSend(int8_T,int8_T);

static void mdlInitializeSizes(SimStruct *S)
{

    ssSetNumSFcnParams(S, NPARAMS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))return;

    ssSetNumContStates(S,0);
    ssSetNumDiscStates(S,0);

    if (!ssSetNumInputPorts(S,2)) return;
    ssSetInputPortWidth(S,0,1);
    ssSetInputPortWidth(S,1,1);
    ssSetInputPortDirectFeedThrough(S,0,0);
```

```
    ssSetInputPortDirectFeedThrough(S,1,0);

    if (!ssSetNumOutputPorts(S,0)) return;

    ssSetInputPortDataType(S, 0, SS_INT8);
    ssSetInputPortDataType(S, 1, SS_INT8);

    ssSetNumSampleTimes(S,1);
    ssSetNumRWork(S,0);
    ssSetNumIWork(S,0);
    ssSetNumPWork(S,0);
    ssSetNumModes(S,0);
    ssSetNumNonsampledZCs(S,0);
}

static void mdlInitializeSampleTimes(SimStruct *S)
{
    ssSetSampleTime(S,0, (real_T) (*(mxGetPr(TS2))));
    ssSetOffsetTime(S,0,0.0);
}

static void mdlOutputs(SimStruct *S, int_T tid)
{
      InputPtrsType      uLeft     = ssGetInputPortSignalPtrs(S,0);
      InputPtrsType      uRight    = ssGetInputPortSignalPtrs(S,1);
      InputInt8PtrsType  uLeftInt8 = (InputInt8PtrsType)uLeft;
      InputInt8PtrsType  uRightInt8 = (InputInt8PtrsType)uRight;

#ifndef MATLAB_MEX_FILE
        TRILOSend(*uLeftInt8[0],*uRightInt8[0]);
#endif
}


#ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

E.4 s_trilo_out.c

```
#define S_FUNCTION_LEVEL 2
#define S_FUNCTION_NAME  s_trilo_out

#include "simstruc.h"
#include <string.h>
#include <windows.h>
#include <stdio.h>
#include <stdlib.h>

#define BAUD  ssGetSFcnParam(S, 0)
#define TS    ssGetSFcnParam(S, 1)
#define LOG   ssGetSFcnParam(S, 2)
#define COM   ssGetSFcnParam(S, 3)
#define DELAY ssGetSFcnParam(S, 4)

#define NSTATES     0
#define NINPUTS     0
#define NOUTPUTS    2
#define NPARAMS     5

extern void TRILOStart(void);
extern void TRILOOpen(int_T, int_T, int_T, int_T);
extern void TRILOLeft(double*);
```

```
extern void TRILORight(double*);
extern void TRILOTerm(void);

static void mdlInitializeSizes(SimStruct *S)
{

#ifndef  MATLAB_MEX_FILE

    int_T Baud, Log, Com, Delay;
    real_T Ts;
    int_T Baud_Range[4]={1200,4800,9600,19200};

    Baud = (int_T) (*(mxGetPr(BAUD)));
    Log  = (int_T) (*(mxGetPr(LOG)));
    Com  = (int_T) (*(mxGetPr(COM)));
    Ts   = (real_T) (*(mxGetPr(TS)));
    Delay= (int_T) (*(mxGetPr(DELAY)));

    printf("\n\n##############################################################################################\n");
    printf("Trilobot Configuration Settings\t:\n\n");
    printf("Selected Communication port\t:\tCOM%i\n",Com);
    printf("Selected Console BaudRate\t:\t%i Baud\n",Baud_Range[Baud-1]);
    printf("Selected block sample-time\t:\t%.4f  [%.0f Hz]\n",Ts,1/Ts);
    printf("Selected additional delay\t:\t%i msecs [%.0f percent of sample time]\n",
            Delay,(((double)Delay/1000)/Ts)*100);

    if(Log == 1)
    {
        printf("Real-time data logging\t\t:\tenabled\n");
    }
    else{printf("Real-time data logging\t\t:\tdisabled\n");}
    printf("\n\n##############################################################################################\n");
    Sleep(1000);

    printf("\nInitializing communication with Trilobot\n");

    TRILOOpen(Baud,Com,Log,Delay);

    printf("Ready to experiment !!!\n");

    TRILOStart();

#endif

    ssSetNumSFcnParams(S, NPARAMS);
    if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S))return;

    ssSetNumContStates(S, 0);
    ssSetNumDiscStates(S, 0);

    if (!ssSetNumInputPorts(S, 0)) return;

    if (!ssSetNumOutputPorts(S, 2)) return;
    ssSetOutputPortWidth(S, 0, 1);
    ssSetOutputPortWidth(S, 1, 1);

    ssSetNumSampleTimes(S, 1);
    ssSetNumRWork(S, 0);
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);

    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
```

```
static void mdlInitializeSampleTimes(SimStruct *S)
{

    ssSetSampleTime(S, 0, (real_T) (*(mxGetPr(TS))));
    ssSetOffsetTime(S, 0, 0.0);
}

static void mdlOutputs(SimStruct *S, int_T tid)
{
    double left;
    real_T *leftlong = ssGetOutputPortRealSignal(S,0);
    double right;
    real_T *rightlong = ssGetOutputPortRealSignal(S,1);

#ifndef  MATLAB_MEX_FILE
    TRILOLeft(&left);
    leftlong[0] = left;
    TRILORight(&right);
    rightlong[0] = right;
#endif
}

static void mdlTerminate(SimStruct *S)
{
    #ifndef  MATLAB_MEX_FILE
        TRILOTerm();
        sleep(1000);
        printf("\nTRILO system disconnected.\n");
    #endif
}

#ifdef  MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

Appendix F

Trilobot 8052 user-programs

In this appendix the code-listing is given of the two user-programs discussed in chapter 4. Both user-programs are written in Assembly language for the 8052 core instruction-set [25, 11, 31]. In these user-programs, some of the standard routines available in the standard software of Trilobot are invoked. These routines are listed in the jump-table given in `trilodef.asm` [2]. These programs are assembled using the Raisonance assembler available in the Raisonance Integrated Development Environment [23].

F.1 Single wheel user program

```
$include(trilodef.a51)
$include(REG51.inc)

dCycle      equ 30h
dCycleC     equ 31h
Flag        equ 32h
SendStat    equ 33h
baud_config equ 38h
baud_double equ 39h

            org 0100h
check_ser : push acc
            clr TR0
            jnb RI, ex_ser
            clr RI
            mov A, SBUF
            mov dCycleC, A
            cpl A
            mov dCycle, A
            pop acc
            setb TR0
            reti
ex_ser:     pop acc
            setb TR0
            ;setb TR2
            reti

            org 1000h
            clr TR0
            push ACC
            jb Flag, motoroff
            setb P3.4
```

```
            mov TH0, dCycle
            mov TL0, dCycle
            setb Flag
            pop ACC
            setb TR0
            reti
motoroff:   clr P3.4
            mov TH0, dCycleC
            mov TL0, dCycleC
            clr Flag
            pop ACC
            setb TR0
            reti


            org 3000h
            lcall trmclr
            clr P3.4
            lcall check_baud
            lcall baud_key
pause:      clr TR2
            lcall utlhloff
            lcall LCD_start
            lcall wait_key
            lcall init
            lcall trmclr
            setb EA
            setb TR0
            setb TR2
            ljmp start

start:      lcall check_key
            jnb TF2,loop
            jb SendStat, send
            clr TF2
            setb SendStat
            jmp start
send:       lcall utlgod
            mov SBUF, DPL
            jnb TI,$
            clr SendStat
            clr TI
            clr TF2
loop:       jmp   start

init:       mov dptr, #0024h
            mov A, #01h
            movx @dptr, A
            mov dptr, #0025h
            mov A, #00h
            movx @dptr, A
            mov SCON, #01010000b
            mov PCON, baud_double
            mov TMOD, #00100010b
            mov TL1,  baud_config
            mov TH1,  baud_config
            setb ES
            setb TR1

            mov dCycle, #0F8h
            mov A, dCycle
            cpl A
            mov dCycleC, A
            clr Flag

            mov dptr, #000Bh
```

```
            mov A, #02h
            movx @dptr, A
            mov dptr, #000Ch
            mov A, #10h
            movx @dptr, A
            mov dptr, #000Dh
            mov A, #00h
            movx @dptr, A

            mov  TH0, dCycle
            mov  TL0, dCycle
            setb ET0
            mov IP, #00000000b
            setb PT0

            mov RCAP2H, #4Ch
            mov RCAP2L, #01h
            mov TH2, #4Ch
            mov TL2, #01h
            clr SendStat

            mov T2CON, #00h
            clr IE.5

            ret



wait_key:   lcall trmgs
wait_Ykey:  cjne  A,#'Y',wait_key
            ret


LCD_start:  lcall trmclr
            lcall   trmpsi
            db  0
            db  'Trilobot  Single'
            db  0
            lcall   trmpsi
            db  40h
            db  'Press Y to Start'
            db 0
            ret

check_baud: lcall utlgd
baud_1:     cjne A, #00000001b, baud_2
            mov baud_config, #-24
            mov baud_double, #00000000b
            lcall trmclr
            lcall   trmpsi
            db  0
            db  'Baudrate  1200'
            db  0
            lcall   trmpsi
            db  40h
            db  'N=reset   Y=GO'
            db 0
            jmp ex_baud
baud_2:     cjne A, #00000010b, baud_3
            mov baud_config, #-6
            mov baud_double, #00000000b
            lcall trmclr
            lcall   trmpsi
            db  0
            db  'Baudrate  4800'
            db  0
```

```
            lcall   trmpsi
            db  40h
            db  'N=reset   Y=GO'
            db 0
            jmp ex_baud
baud_3:     cjne A, #00000100b, baud_4
            mov baud_config, #-3
            mov baud_double, #00000000b
            lcall trmclr
            lcall   trmpsi
            db  0
            db  'Baudrate  9600'
            db  0
            lcall   trmpsi
            db  40h
            db  'N=reset   Y=GO'
            db 0
            jmp ex_baud
baud_4:     cjne A, #00001000b, baud_s
            mov baud_config, #-3
            mov baud_double, #10000000b
            lcall trmclr
            lcall   trmpsi
            db  0
            db  'Baudrate 19200'
            db  0
            lcall   trmpsi
            db  40h
            db  'N=reset   Y=GO'
            db 0
            jmp ex_baud
buad_s:     mov baud_config, #-3
            mov baud_double, #00000000b
            lcall trmclr
            lcall   trmpsi
            db  0
            db  'Baudrate 9600D'
            db  0
            lcall   trmpsi
            db  40h
            db  'N=reset   Y=GO'
            db 0
ex_baud:    ret


check_key:  lcall trmgs
            jnc ex_keys
check_N:    cjne    A,#'N',check_Y
            lcall trmclr
            lcall sysres
check_Y:    cjne  A,#'Y',ex_keys
            ljmp pause
            lcall trmclr
ex_keys:    ret

baud_key:   lcall trmgs
            jnc baud_key
baud_N:     cjne    A,#'N',baud_Y
            lcall check_baud
baud_Y:     cjne  A,#'Y',baud_key
ex_bkeys:   ret
            end
```

F.2 Two wheel user program

```
$include(trilodef.a51)
$include(REG51.inc)

dCycle1      equ 30h
dCycle2      equ 31h
Cycle1       equ 32h
Cycle2       equ 33h
Cycle3       equ 34h
snddef       equ 35h
sndstat      equ 36h
status       equ 37h
baud_config  equ 38h
baud_double  equ 39h

             org 0100h
check_ser :  clr TR2
             push acc
             jnb RI, check_TIn
             clr RI
             mov A, SBUF
             jb acc.7, is_neg
             mov dCycle2, A
             jmp recomp
is_neg:      mov dCycle1, A
recomp:      mov A, dCycle2
             add A, dCycle1
             jb acc.7, neg_sign
             clr sign
             mov Cycle2, A
             mov A, dCycle1
             cpl A
             inc A
             mov Cycle1, A
             mov A, dCycle2
             cpl A
             mov Cycle3,A
             pop acc
             setb TR2
             reti
neg_sign:    setb sign
             cpl A
             inc A
             mov Cycle2, A
             mov A, dCycle2
             mov Cycle1, A
             mov A, dCycle1
             cpl A
             inc A
             cpl A
             mov Cycle3, A
             pop acc
             setb TR2
             reti
check_TIn:   jnb TI, ex_ser
ex_ser     : pop acc
             setb TR2
             reti

             org 1000h
             clr TR2
             push ACC
             mov A, status
state1:      cjne A, #00h, state2
             lcall utlhlon
```

```
            lcall utllaon
            mov A, Cycle1
            mov B, #02h
            mul AB
            cpl A
            mov B, #10h
            mul AB
            mov TL2, A
            mov A, B
            add A, #0F0h
            mov TH2, A
            mov status, #01h
            clr TF2
            pop ACC
            setb TR2
            reti
state2:     cjne A, #01h, state3
            jb sign, clr_m2
            lcall utlhloff
            jmp exit2
clr_m2:     lcall utllaoff
exit2:      mov A, Cycle2
            mov B, #02h
            mul AB
            cpl A
            mov B, #10h
            mul AB
            mov TL2, A
            mov A, B
            add A, #0F0h
            mov TH2, A
            mov status, #02h
            clr TF2
            pop ACC
            setb TR2
            reti
state3:     cjne A, #02h, return
            jb sign, clr_m1
            lcall utllaoff
            jmp exit3
clr_m1:     lcall utlhloff
exit3:      mov A, Cycle3
            mov B, #02h
            mul AB
            cpl A
            mov B, #10h
            mul AB
            mov TL2, A
            mov A, B
            add A, #0F0h
            mov TH2, A
            mov status, #00h
            clr TF2
            djnz sndstat, nosample
            setb sndtrig
            mov sndstat, snddef
nosample:   pop ACC
            setb TR2
            reti
return:     pop ACC
            setb TR2
            reti

            org 3000h
            lcall trmclr
            lcall check_baud
```

```
            lcall baud_key
pause:      clr TR2
            lcall utllaoff
            lcall utlhloff
            lcall LCD_start
            lcall wait_key
            lcall init
            lcall init_enc
            lcall trmclr
            ljmp start

start:      lcall check_key
            jnb sndtrig, loop
            lcall get_enc
            clr sndtrig
loop:       jmp   start

get_enc:    mov A, TL0
            lcall sndchar
            lcall utlgod
            mov A, DPL
            lcall sndchar
            ret

wait_key:   lcall trmgs
wait_Ykey:  cjne  A,#'Y',wait_key
            ret

LCD_start:  lcall trmclr
            lcall   trmpsi
            db  0
            db  'TU/e    Trilobot'
            db  0
            lcall   trmpsi
            db  40h
            db  'Press Y to Start'
            db 0
            ret

init:       mov dptr, #0024h
            mov A, #01h
            movx @dptr, A
            mov dptr, #0025h
            mov A, #00h
            movx @dptr, A
            mov SCON, #01010000b
            mov PCON, baud_double
            mov TMOD, #00100101b
            mov TL1,  baud_config
            mov TH1,  baud_config
            setb ES
            setb TR1
            mov dCycle1, #-1
            mov dCycle2, #2h
            mov Cycle1, #1h
            mov Cycle2, #1h
            mov Cycle3, #7Dh
            mov status, #00h
            mov snddef, #18
            mov sndstat, snddef
            clr sign
            mov dptr, #002Bh
            mov A, #02h
            movx @dptr, A
            mov dptr, #002Ch
            mov A, #10h
```

```
            movx @dptr, A
            mov dptr, #002Dh
            mov A, #00h
            movx @dptr, A
            clr RCLK
            clr TCLK
            clr EXEN2
            clr RL2
            setb IE.5
            mov RCAP2H, #0FFh
            mov RCAP2L, #000h
            setb TR2
            setb EA
            mov IP, 00000000b
            ret

init_enc:   mov TH0, #0
            mov TL0, #0
            clr ET0
            clr P3.4
            ret

sndchar:    clr TI
            mov SBUF, A
txloop:     jnb TI, txloop
            clr TI
            ret

check_baud: lcall utlgd
baud_1:     cjne A, #00000001b, baud_2
            mov baud_config, #-24
            mov baud_double, #00000000b
            lcall trmclr
            lcall   trmpsi
            db  0
            db  'Baudrate  1200'
            db  0
            lcall   trmpsi
            db  40h
            db  'N=reset   Y=GO'
            db 0
            jmp ex_baud
baud_2:     cjne A, #00000010b, baud_3
            mov baud_config, #-6
            mov baud_double, #00000000b
            lcall trmclr
            lcall   trmpsi
            db  0
            db  'Baudrate  4800'
            db  0
            lcall   trmpsi
            db  40h
            db  'N=reset   Y=GO'
            db 0
            jmp ex_baud
baud_3:     cjne A, #00000100b, baud_4
            mov baud_config, #-3
            mov baud_double, #00000000b
            lcall trmclr
            lcall   trmpsi
            db  0
            db  'Baudrate  9600'
            db  0
            lcall   trmpsi
            db  40h
            db  'N=reset   Y=GO'
```

```
            db 0
            jmp ex_baud
baud_4:     cjne A, #00001000b, baud_s
            mov baud_config, #-3
            mov baud_double, #10000000b
            lcall trmclr
            lcall   trmpsi
            db  0
            db  'Baudrate 19200'
            db  0
            lcall   trmpsi
            db  40h
            db  'N=reset   Y=GO'
            db 0
            jmp ex_baud
baud_s:     mov baud_config, #-3
            mov baud_double, #00000000b
            lcall trmclr
            lcall   trmpsi
            db  0
            db  'Baudrate 9600D'
            db  0
            lcall   trmpsi
            db  40h
            db  'N=reset   Y=GO'
            db 0
ex_baud:    ret

check_key:  lcall trmgs
            jnc ex_keys
check_N:    cjne    A,#'N',check_Y
            lcall trmclr
            lcall sysres
check_Y:    cjne  A,#'Y',ex_keys
            ljmp pause
            lcall trmclr
ex_keys:    ret

baud_key:   lcall trmgs
            jnc baud_key
baud_N:     cjne    A,#'N',baud_Y
            call check_baud
baud_Y:     cjne  A,#'Y',baud_key
ex_bkeys:   ret

            end
```