Techniques for the implementation of control algorithms using low-cost embedded systems

Thesis submitted for the degree of

Doctor of Philosophy

at the University of Leicester

By

Ricardo Bautista Quintero
BEng., MSc

Embedded System Laboratory

Department of Engineering

December 2008

Techniques for the implementation of control algorithms using low-cost embedded systems

Ricardo Bautista Quintero

Abstract

The feedback control literature has reported success in numerous implementations of systems that employ state-of-the-art components. In such systems, the quality of computer controller, actuators and sensors are largely unaffected by nonlinear effects, external disturbances and finite precision of the digital computer. Overall, this type of control systems can be designed and implemented with comparative ease. By contrast, in cases when the implementation is based on limited resources, such as, low-cost computer hardware along with simple actuators and sensors, there are significant challenges for the developer.

This thesis has the goal of simplifying the design of mechatronic systems implemented using low-cost hardware. This approach involves design techniques that enhance the links between feedback control algorithms (in theory) and reliable real-time implementation (in practice).

The outcome of this research provides a part of a framework that can be used to design and implement efficient control algorithms for resource-constrained embedded computers.

The scope of the thesis is limited to situations where 1) the computer hardware has limited memory and CPU performance; 2) sensor-related uncertainties may affect the stability of the plant and 3) unmodelled dynamic of actuator(s) limit the performance of the plant.

The thesis concludes by emphasising the importance of finding mechanisms to integrate low-cost components with nontrivial robust control algorithms in order to satisfy multi-objective requirements simultaneously.

Acknowledgement

Firstly, I would like to express my appreciation to my supervisor Dr. Michael Pont for his guidance throughout this research project.

Next, I would like to thank to the Mexican Government (Coordinación Sectorial del Desarrollo Académico) for awarding a full-time scholarship. In addition, I would like to acknowledge to my colleges of the Centro Nacional de Actualización Docente for their support over these years of research.

I also would like to thank to my colleges, Ayman G. Adi M., Ali I., Ali N., Azura, A., Farah, L., Keith A. Chisanga M., Dong L., Huiyan W., Kam C., Mouaaz N., Musharaf H., Muhammad A., Susan K., Teera P., Peter V., Zemian H. All of them are my friends from Embedded System Laboratory. I am gratefully for all their help and support, especially Dr. Devaraj Ayavoo who encouraged me to keep ahead in this hard working project.

A special acknowledge to Dr. Fernando Schlindwein and George Loukadaki who help me with the proofreading of this thesis.

My greatest support comes from my beloved wife. She played a truly instrumental role on seeing this project to completion. Thank you for all your help in this challenging stage, I may not be able to express enough gratitude in my whole life.

I also would like to thank my children, my little Ricardito, who has made me the happiest Dad in the world. His joy is the motivation to do my best. My little Yaretzi you just came to this world one day after the submission of this thesis, thank you for your patience, and remember, “You will be loved for ever”.

Finally yet importantly, thank my parents, Maria del Carmen Quintero Ruiz and Enrique Bautista Garces, my sisters Chayo and Carmelita, my brothers, Felipe and Victor.
I cannot thank enough for their constant support over the last few years. Thank you for always being there for me.

I dedicate this work to my beloved family

Sonia Maribel Garcia-Garcia,
Ricardo Bautista-Garcia
Yaretzi Bautista-Garcia

Table of contents

List of figures

List of publications

Directly-related publications

A number of papers have been published during the course of the work described in this thesis. These are listed below. Please note that the contents of some of these papers have been adapted for presentation in this thesis: where applicable, a footnote at the beginning of a chapter indicates that material from one or more papers has been included.

Bautista-Quintero R. and Pont, M. J. (in preparation) "Supporting the use of auto-code generation tools in feedback control implementations using resource-constrained embedded systems". Transactions of the Institute of Measurement and Control.

Bautista-Quintero, R. and Pont, M. J. "Implementation of H-infinity control algorithms for sensor-constrained mechatronic systems using low-cost microcontrollers" in press, IEEE Transactions on Industrial Informatics, Vol. 4, issue 3, Aug. 2008, Pages 175-184.

Bautista-Quintero, R. and Pont, M. J. "Is fuzzy logic a practical choice in resource-constrained embedded control systems implemented using general-purpose microcontrollers?" Proceedings of the 9th IEEE International Workshop on Advanced Motion Control. Vol. 2, March 2006, Pages 692-697.

Bautista-Quintero R. and Pont, M. J. "The application of fuzzy logic control in resource-constrained embedded systems". Poster presented at the Postgraduate Festival, University of Leicester, May 2006.

Bautista-Quintero, R., Pont, M. J. and Edwards T. "Comparing the performance and resource requirements of "PID" and "LQR" algorithms when used in a practical embedded control system: A pilot study", presented in the Embedded System Forum, Birmingham UK, Oct. 2005, pages 262-289, Published by the University of Newcastle.

Associated publication

Pont, M.J., Kurian, S. and Bautista-Quintero, R. "Meeting real-time constraints using 'Sandwich Delays". In press on Transactions on Pattern Languages of Programming.

List of Abbreviations, Symbols and Units

Abbreviations

ACCS	Adaptive Cruise-Control System
ADC	Analogue to Digital Converter
CASE	Computer Aided Software Engineering
CPLD	Complex Programmable Logic Device
DAC	Digital to Analogue Converter
DC	Direct Current
DSP	Digital Signal Processor
ESL	Embedded Systems Laboratory
ECSs	Embedded Control Systems
FBD	Function Block Diagram
FLC	Fuzzy Logic Control
FPGA	Field Programmable Gate Array
FWR	Finite Word Length
HIL	Hardware-in-the-Loop
I/O	Input/Output
IL	Structured List
IMP	Implementation
Kbps	kilo bits per second
LCD	Liquid Crystal Display
LED	Light Emitting Diode
MF	Membership Function
MIMO	Multiple input multiple output
MPC	Model Predictive Control
H∞	H infinity
LQR	Linear Quadratic Regulator
LD	Ladder Diagram
SISO	Single Input Single Output
SIMO	Single Input Multiple Output
SFC	Sequential Function Chart
SMC	Slide Mode Control
ST	Structured Text
PID	Proportional Integral Derivative
PLD	Programmable Logic Device
ppr	Pulses Per Revolution
PTTES	Patterns for Time-Triggered Embedded Systems
PWM	Pulse Width Modulation
UML	Unified Modelling Language

Symbols

a	Acceleration
Fr	Frictional coefficient
m	Mass
v_f	Final speed
v_i	Initial speed
Δx	Displacement

Units

kg	Kilogram
m/s	Metres / Second
ms	Millisecond
s	Second
µs	Microsecond

Chapter 1
Introduction

This chapter explains the context, relevance, aims, and contributions of the research project which is described in this thesis.

1.1 Introduction

Feedback control is one of the most astonishing mechanisms found in nature. According to Charles Darwin's theory, feedback control is the self-regulative process that preserves life through evolution [Lewis, 1992]. This mechanism, by which a dynamic system maintains equilibrium (by monitoring its own outputs and modifying appropriate inputs) has been widely studied.

Construction of systems based on these self-regulative principles – more commonly known as automatic control [Franklin et al., 1993] – has a long history. However, the early development stage of such systems might suggest a lack of the use of complex theoretical tools. For example, although the seminal work of James Watt on the fly-ball governor was completed in the 1760s [Bishop, 2006] (and is perhaps the first self-regulative machine of the modern era), no mathematical analysis of the machine's dynamics was published at that time: development and performance improvements of this machine were based on purely experimental work. It was not until a century later that James C. Maxwell proposed the first non-trivial dynamic model of the fly-ball governor [Maxwell, 1868].

This mismatch in time between the practical invention and analytical description might be the first evidence of the gap of knowledge between theory and practice in the field of feedback control. This appears to have occurred because the central

ideas of feedback control in practice were developed separately from the theoretical perspective.
While control theory grew as part of applied mathematics [Bernstein, 1998], the implementation of feedback control – in general – combines multi-disciplinary techniques [Joshi, 1999].

Today, although the control theory has matured and evolves slowly in comparison to practice, there are still multiple impediments to bridge the gap between the two [Bernstein, 1999]. This also entails important research and development challenges.

1.2 Bridging the theory-practice gap in control systems

For non-trivial control applications, bridging the gap has gained enormous relevance. A proper integration of theoretical ideas with practical techniques increases significantly the performance of products and processes. Therefore, products and processes can be more competitive and profitable [Bernstein, 2002].

The development of technology over the last century facilitated the implementation of control ideas using conventional systems such as: mechanical devices and electro-mechanical tools [Minorsky, 1922]; pneumatics [Bennett, 1993]; hardwired circuits [Gill, 1998] and analog electronic components [Black, 1984]. Nevertheless, most of this conventional technology used for automatic control was replaced by the use of digital computers [Donaghey, 1976], [Wang, 1981]. The re-programmability feature of such devices enhances the flexibility and adaptability in the control system's design. For those reasons, computers have been credited as the most influential tool in the systematic integration of control theory with practical implementation [Bahram and Hassul, 1993].

In this context, electronics contributed enormously in the invention of digital computers. Transistor development, in particular, made such computers more

compact and reliable than previous designs (e.g. those based on vacuum tubes). Transistor integration in micro-chips was the key factor for advances in computer design. Development has progressed from small, medium, large, very-large, ultra-large, wafer-scale of integration (SSI, MSI, LSI, VLSI, ULSI), to more recent system-on-chip (SoC) and three-dimensional integrated circuits (3D-ICs). The result of all this progress in technology has allowed an efficient combination of power and affordability of computer systems.

Among various digital computer-hardware designs, one category has grown enormously. This category is called embedded computer systems, or simply "embedded systems". Many applications of feedback control using embedded systems (embedded control systems) have contributed to the challenging task of understanding the theory-practice interaction.

1.3 Background of embedded systems

Although the meaning of embedded systems is unfamiliar in a social context, a modern society has eagerly bought products based on this technology. Many everyday products, like MP3 players, mobile phones, cars' computers, electronic medical equipment and many more are associated with the embedded system world. The variety of such products is virtually limitless. In fact, 98 percent of the digital programmable devices produced nowadays belong to the embedded system category [Gaetano and Roy, 2000]. In 2004, there were three embedded systems per person on Earth [WSTS, 2005] and indicators predict an exponential growth for the coming years [Fisher et al., 2004].

1.3.1 Definition

Several influential definitions of embedded systems have been suggested from different authors [Butazzo, 2002], [Pont, 2001], [Laplante, 2005] and more recently [Apneseth, 2006]. However, in 1990 the IEEE standard glossary of software engineering terminology [Standards Coordinating Committee of the

IEEE Computer Society, 1990], defined embedded system as *"a computer system that is part of a larger system and performs some of the requirements of that system"*. In fact, an embedded computer is designed for special-purpose applications [Ganssle and Barr, 2003]. In a wider and multidisciplinary context, an embedded system is not only a computer, but also hardware (electronic, mechanical structures, etc.) designed for a specific application [Henkel et al., 2003]. In this thesis, the kind of embedded systems studied are such related to control applications including the electronic interface, actuators and sensors.

1.3.2 Origins of embedded control systems

The history of embedded control systems (ECSs) dates back to 1961, when the first special-purpose computer, Autonetics D-17 (see Figure 1-1), was used for the US missile guidance systems [Lin, 2003]. Based on the same technology, in 1964, the Apollo guidance computer (AGC) was programmed to control the navigation functions of the Apollo spacecraft [Eldon, 1996]. Almost simultaneously, the first automotive embedded-system was employed in the Volkswagen 1600, controlling the engine's fuel injection [Baumann, 1967].

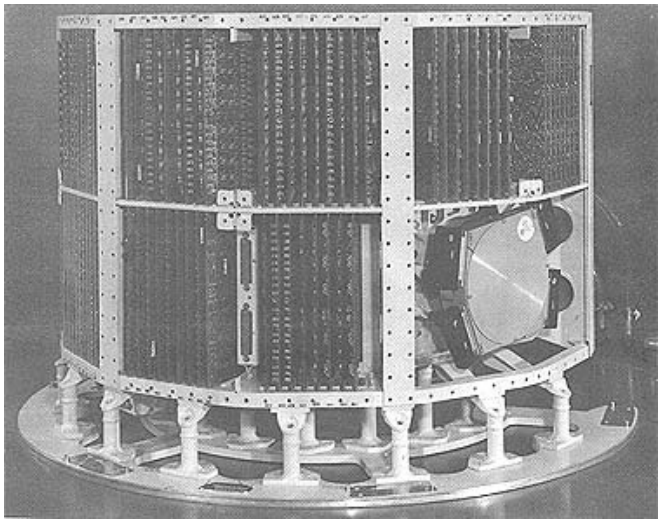

Figure 1-1 Autonetics D-17, the first digital embedded computer used in MINUTEMAN missile guidance systems (Image courtesy of Computer History Museum)

The revolutionary technology of ECSs opened new possibilities in different commercial sectors. According to [WSTS, 2005], the most influential areas in ECSs are: automotive, avionics, telecommunications, consumer electronics, industrial automation and medical products.
The demand of a rapid growth in this field introduced – of course – many challenges in the scientific and technical domain.

1.4 Designing and implementing ECSs

Designing and implementing ECSs can be extremely challenging. This section discusses some of the challenges faced in this field.

1.4.1 Varying hardware platforms

ECSs – in general – offer unique advantages of cost-efficiency, reliability, flexibility and performance [Donaghey, 1976]. This is perhaps because microprocessors hardware has improved based on the well-known Gordon Moore's law (integrated circuits would double in performance every two years approximately) [Moore, 1979]. However, it is becoming much more difficult to develop and integrate effective software for a range of different hardware patforms [Napper, 1998]. This is mainly because embedded hardware is designed according to requirements for specific circumstances. The software tools available are oriented more to desktop applications rather than embedded systems. There is a lack of software support tools for the vast options of new hardware [Napper, 1998].

In some safety-critical applications, unreliable software has led to many human losses [Leveson and Turner, 1993], [Grottke and Trivedi, 2007]. In fact, if the software has an error rate of 0.1 percent, it could lead to 500 exceptions per week in medical surgeries that rely on embedded systems and 18 airplane crashes per day [Balzert, 2000].

In general, the cost of software errors is becoming very high. For instance, 31 companies, reported losses due to software-related issues of about $17,348.450 million, between 1992 and 2005 [Charette, 2005]. All of these errors were entirely preventable [Charette, 2005].

1.4.2 Limitations of traditional control algorithms

Implementing ECSs demands practical methods for customisation in different control applications. One algorithm extensively used in control applications is "proportional-integral-derivative" (PID), which is computationally simple to implement [Åström and Hägglund, 2001], [Bishop and Dorf, 2004], [Ogata, 2002]. In fact, PID is a dominant form of feedback control in industry, [Åström, 1985], [Bennett, 1993], [Donaghey, 1976]: more than 90% of all feedback loops are based on this technique [Åström and Hägglund, 2001].

Despite its polularity, the PID algorithm is incapable of coping with some control problems, see for instance [Atherton and Majhi, 1999], [Dicheng et al., 2004]; the system performance using a PID control often depends of some specific parameters of the system, that constrains appropriate control tuning [Bekit et al., 1998]. In addition, some tuning methods are performed on-line, which limits implementation of unstable open-loop systems [Atherton and Majhi, 1999]; [Kawaji and Kanazawa, 1991]. For multivariable systems where the dynamics are complex, it is rarely possible to find an efficient PID solution which will provide good performance [Atherton and Majhi, 1999], [Machado and Galhano, 1995], [Tang et al., 2001].

Limitations of the traditional techniques to cope with real control problems have motivated the invention of new forms of control in order to improve performance; see for instance, [Doyle et al., 1989], [Zadeh, 1965], [Maciejowski, 2000], [Tang et al., 2001] and [Lewis, 2005]. Some of these approaches were developed in the last few decades in order to be robust against uncertainties found in actual

implementations (where there is not a straightforward mechanisim to identify and quantify uncertainties using numerical simulations). However these modern approaches are largely irrelevant for today's requirements. This is due to different factors, for instance:

1) The conservatism of industry to use classic and off-the-shelf methods [Auslander et al., 1978].
2) The short life-cycle of today's products, that constrains extended research and development programs [Schlett, 1998].

The complexity of integrating advanced forms of control algorithms – usually more sophisticated in terms of computer requirements – into common practices presents additional challenges. Moreover, the tools available for support of more advanced algorithms are used mainly in research [Skogestad and Postlethwaite, 2005], and rarely in industry [Åström and Hägglund, 2001].

In addition to the lack of support for these modern control techniques, the implementation of control algorithms requires an efficient and reliable software architecture platform[1]. For low-cost computers, multiple trade-offs have to be studied. For instance, the reliability of the execution sequence of the control tasks can improve the robustness against failure. However, this can limit the flexibility and adaptability for multiple scenarios (uncertainties caused for acquisition, algorithm processing or actuation).

1.4.3 Low-cost ECSs

One of the main factors to consider in embedded systems design is the cost (see [Slomka et al., 2000] and [Napper, 1998]). Such cost constraints can be a factor that limits achieving of the desired performance. For example, many aerospace and military control systems employ state-of-the-art sensors, actuators and

[1] In this context, "software architecture platform" is the program that manages the execution sequence of the application routines.

computer systems. Such systems are accurate, predictable and efficient. By contrast, many commercial applications face severe cost constraints and must be implemented using simpler and cheaper sensors and actuators along with limited computer power (low memory and CPU performance): see for instance, [Henriksson, 2006], [Li and Meijer, 1998], [Kaul et al., 1997] and [Rauta et al., 1996].

Different implementation approaches are required for these "resource rich" and "resource constrained" system designs: see the the computational contrast between [Campbell, 1999] and [Wang and Yang, 2004].

For resource-constrained systems, detailed mathematical analysis is essential to represent accurately the dynamics of such systems. However the inclusion of low-cost hardware (sensors, actuators and controllers) leads to complex models which are rarely studied in the educational literature [Bishop and Dorf, 2004], [Ogata, 2002], [Franklin et al., 1993]. Even for specialised scientific literature, neglecting the dynamics of actuators and sensors is a common practice [Kawaji and Kanazawa, 1991], [Spong, 1987], [Sun and Meng, 2004]. For that reason, success in the implementation (integration of software with hardware) of control algorithms in situations where there is large uncertainty over the system's dynamics still requires a long and tedious process that is prone to errors. These challenges are compounded when the control system is to be implemented using a low-cost computer platform which has severe resource constraints compared to a modern "desktop computer".

1.5 Aims of the project described in this thesis

The project described in this thesis suggests that, when "ideal" system components are used in a resource-rich design it may be possible to deal with simple theory-based models for implementation purposes. However, when

dealing with low-cost system components, it is rarely possible to find a straightforward mechanism to implement an effective controller directly.

1.5.1 Specific problem studied in this thesis

There are well stablished computer control implementation issues previously studied in the literature of this field, see for example [Forsythe and Goodall, 1991], [Williamson, 1991] and [Istepanian and Whidborne, 2001]. In general control implementation nowadays deals with problems related to limitations in actuators, sensors and digital computers. Non-linearities in the response of actuators, noise, rate limits such torque and bandwidth. In sensors the inevitable delay response, range quantifications errors, sampling rate. Particularly in the field of embedded computer quantification of jitter computation time, loading, variables wordlenghts, precision, numerical conditioning, round-off errors, memory, concurrencies, fix-point and floating point considerations, ADC sampling time, PWM frequencies and precision, etc.

Amoung the universe of discussion of all these factors, the specific problems studied in this thesis are focused on finding efficient implementation techniques for uncertainties in the sensor (low-resolution) and actuator (unknown dynamic) along limited memory and CPU performance. Consideration of the wordlengh of the variables are taken into acount in order to reduce the computational load and memory requirements.

In this context, the thesis explores the implementation of modern control algorithms for mechatronic systems using low-cost embedded systems. The techniques used to support the design of reliable controllers are investigated in situations where:

1) The design is implemented using a resource-constrained computer.

2) Sensor limitations and their effects on affecting the overall performance or stability.
3) There are restrictions in the actuator system.

The initial phase of this research explores the implementation of classic control algorithms (as a benchmark) in different scenarios that entail restrictions due to the low-cost components. The second phase of the research investigates modern and intelligent control algorithms and the feasibility of their implementation in resource-constrained computers.

1.6 Key contributions

This work presents three important contributions to this research area:

First, this thesis shows the techniques for implementing modern control structures based on an optimal LQR using a very low-cost embedded microcontroller.

Second, the thesis demonstrates the importance of characterising actuator dynamics when the overall system employs low-cost components. Theoretical and practical aspects in this context are suggested in order to facilitate the design of custom applications. PID and fuzzy logic control (FLC) algorithms are used to illustrate the experimental validation of this aspect.

Third, the research introduces a novel method which is intended to assist in the design and implementation of optimal H-infinity (H_∞) algorithms in low-cost mechatronic applications. The problem considered here is a position control system in a situation where there are both sensor-related uncertainties (caused by low–resolution sensors) and limited computational resources. A model of how the uncertainties can be seen from the low computational resources controller is proposed. Additionally the impact in memory reduction and CPU load is measured when a proper realisation of the dynamic control is chosen.

1.7 Thesis layout

Following this introduction, Chapter 2 presents the literature review that surveys the research and development of low-cost embedded systems. Chapter 3 describes the test-bed used in this project and practical design and benchmark control implementation based on the classic PID. Chapter 4 shows a case study of implementating a controller for intelligent control (Fuzzy logic). Chapter 5 explores the implementation of the optimal linear quadratic regulator (LQR). Chapter 6 explores the use of robust control (H-infinity) in computers with limited resources and sensors with low resolution used in typical mechatronic applications. Discussions are presented in Chapter 7. General conclusions are presented in Chapter 8. Appendix A contains the source code of the control programs used for this research. Appendix B includes the paper Meeting Real-Time Constraints Using "Sandwich Delays".

1.8 Conclusion

This introduction chapter has presented a summary of the importance of the gap between the theory and practice in control systems design. In particular, it showed the role played by the computer technology in bridging this gap. Based on the issues presented, embedded systems used for control applications appeared as the key element in the understanding between the theoretical control systems design and their real-time implementation. Special attention was drawn to the situations when the system has important limitations in hardware resources.

The remainder of this thesis will describe the work undertaken throughout this project.

Chapter 2
Literature review of ECSs

This chapter provides a review of previously published work in the area of ECSs.

First, it considers the historical evolution of embedded control hardware and the algorithms reported for different applications. Second, the importance of reducing cost in the design and implementation of embedded systems is considered. Over the last few decades, this economical factor has had an enormous influence in the research and development of control systems. Additionally, this chapter surveys the current state-of-the-art developments in ECSs and future trends.

2.1 Introduction

Embedded systems and computer science share some common interests and history. In fact, according to [Sangiovanni-Vincentelli and Pinto, 2005] the core of embedded systems emerged as the conjunction between engineering and computer science, see Figure 2-1. However, they have some very basic differences.

Traditionally computer science deals with general-purpose computer problems; in contrast, embedded systems cope with optimisation of special purpose hardware and software. Research in embedded systems is specialised in –for instance– scheduling[2] algorithms [Liu and Layland, 1973], [Butazzo, 2005], [Cervin, 2003]; power efficiency [Moyer, 2001], [Simunic et al., 2001], [Dongkun et al., 2002];

[2] Scheduler in embedded systems is the policy used to execute multiple computer tasks in order to perform a process

code size reduction [Leupers, 2002], [Pinnepalli et al., 2007] and reliability for safety-critical applications [Fernandes and Maciel, 2003], [Pont and Banner, 2002] and [Short and Pont, 2005].

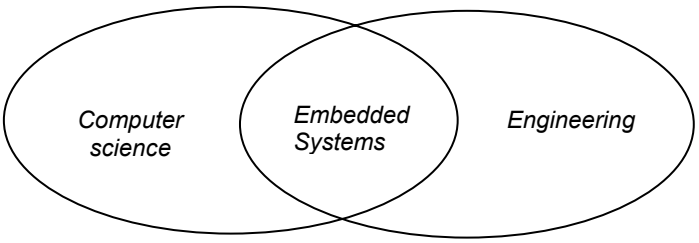

Figure 2-1 Relation between computer science and engineering according to [Sangiovanni-Vincentelli and Pinto, 2005]

Embedded systems research in this thesis deals with the real-time implications of limited computational resources. This condition constrains the execution tasks before maximum execution time is completed, since a missing deadline produces unpredictable responses in the control signal (hard real-time) [Liu and Layland, 1973], [Veysel et al., 2001].

In the field of ECSs real-time synchronisation of execution tasks is generally required. In most control applications the completion of the computer tasks before any deadlines is crucial [Henriksson, 2006]. These challenges impose some hardware requirements in order to achieve specific system performance. Typically the resources required are both enough CPU power and memory capacity [Cervin, 2003].

The development of ECSs has grown exponentially in the last few decades. A summary of the most important achievements in this area is shown next.

2.2 Evolution of ECSs

In the 1960s, general-purpose computers were not a cost-efficient choice for many commercial applications. However, a revolution in the embedded systems world

in the early 1970s brought the ability to integrate a low-cost computer in one chip [Faggin et al., 1996]. This paradigm shift caused a tremendous impact in industrial process control [Auslander et al., 1978] and consequently in many other areas.

Due to the historical progress of the hardware and control algorithms employed in embedded systems, this thesis has divided a survey into four generations (approximately each generation constitutes one decade of evolution) which are presented next.

2.2.1 First generation of ECSs

The history of microprocessors dates back to 1971 when Intel launched the 4004. This device is recognized as the first microprocessor used in embedded applications [Pratt and Brown, 1975], [Goksel et al., 1975]. The 4004 was a 4-bit CPU and had a maximum speed clock of 740 kHz. Using that speed of clock the microprocessor executed 94,000 instructions per second (IPS), which is about 5000 times slower than a modern microprocessor. Nevertheless, that CPU power was enough to perform different computer tasks.

Originally, Intel designed the 4004 for *Busicom*. It was used in the construction of several models of calculators, office-printing machines and cash-registers. However, the designers of the 4004 considered that its primary market was oriented to control devices [Faggin et al., 1996]. In the early stage of the use of this microprocessor, combinatorial and sequential control applications were predominant, see for instance [Pratt and Brown, 1975].

In other control applications, like industrial robots, the very first approach was the *Unimate* (universal automation) system [Spong and Vidyasagar, 1989], patented in 1954. This control system had two non-embedded computers, PDP-10 and PDP-15 [Goksel et al., 1975]. However, this approach was important for

embedded robotics applications, since further developments were based on the same parallel processing architecture. For instance, the microcomputer MCS-4 (which included a 4004 microprocessor) was one of the first embedded systems used for the dynamic control of the robots. This approach is perhaps the first industrial distributed embedded control system. See Figure 2-2.

The robotic system described in [Goksel et al., 1975] had two computers. One was a general-purpose computer performing supervisory work (coordination and planning trajectories, communications, etc.), whereas the second computer was based on embedded processors used for executing the control tasks (dynamic control of links and gripper). See Figure 2-2.

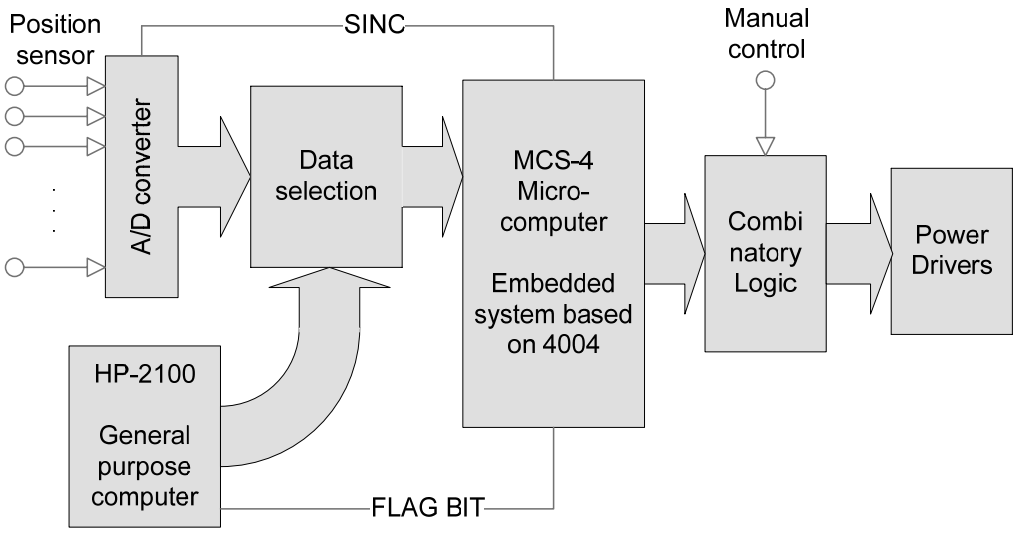

Figure 2-2 Control hardware of an industrial robot arm using two computers: general purpose HP-2100 and the embedded microprocessor 4004. The parallel computer suggests a structure similar to modern embedded distributed control systems. Diagram adapted from [Goksel et al., 1975]

Using multiple computers with different capacities (working together for a common purpose) made possible the extensive development of distributed control systems in this field [Auslander et al., 1978].

Subsequent developments in microprocessors hardware, such as the Intel 8008, (8-bit microprocessor) made possible the implementation of more sophisticated algorithms for automatic control processes. For instance, in the field of industrial machines, the 8008 replaced impractical hard-wired control systems, improving the flexibility and quality of the final products: see [Seim, 1975] and [Ridgeway, 1975].

The rapid growth of microprocessor-based technology made it possible to design special-purpose mini-computers for industrial applications. Embedded computers called "programmable logic controllers" (PLCs) substituted the tedious hard-wire process, commonly practiced in the 1960s [Erickson, 1996]. PLC hardware was commonly designed using embedded systems with enhanced temperature range and noise immunity.

In terms of software, the first language used in PLCs was Assembly, which became obsolete when Ladder language later appeared. This change in embedded-software development was driven by forces of compatibility with older systems (based on hardwired electronic, pneumatic and electromechanical controllers) [Colnaric, 1999]. In the 1970s, the power of Ladder language allowed a familiar direct interface to the user with the common scripts used for hard-wired practices.

2.2.2 Second generation of ECSs

For processing and control of real-time tasks, sophisticated systems with higher microprocessor speed were required [Donaghey, 1976]. Enhancements based on 8-bit technology, allowed the use of embedded microprocessors in a wide range of applications. For instance: in medical equipment [Bronzino, 1982]; enhanced Engine Control Units (ECU) for commercial vehicles [Cuatto et al., 1998]; cost-effective mechatronic machinery used in industry for production lines [Nakamura

et al., 2004]; office and commercial machines such as printers [East, 1984]. In this context, perhaps the most influential area for automatic control implemented with embedded systems was the chemical industry. A wide range of development in this area was based on monitoring and control processes [Donaghey, 1976].

Microcomputers in this generation included embedded system processors, with different types of peripherals such as memory, interface ports, communication systems and displays [East, 1984].

Software developments in embedded systems soon introduced added functionalities in order to improve the development of software [Donaghey, 1976]. For example, special editing facilities such as breakpoints during program debugging, memory address pointers and floating-point-multiplication subroutines. These approaches were merely meant to simplify the program codification. In control algorithms (both sequential and automatic control), the use of these subroutines improved the maintenance of code. These software features were mostly implemented in 8-bit CPU microprocessors, such as the 8008 and 8080 from Intel, the Z80 from Zilog and the 6800 by Motorola Corporation. This hardware appeared as an optimal trade-off between the data bus size and CPU performance, minimizing expensive system support [Donaghey, 1976].

General processes for feedback control and small versions of operating systems were designed for this second generation of microprocessors [Donaghey, 1976]. The basic scheme of the process-control program is shown in Figure 2-3. The fundamental core of this routine was composed of 1) Acquisition; 2) Control algorithm; 3) Actuation. The operating system program was meant to organize the execution of these basic tasks and coordinate with additional functions. These functions included the management of both analogue-to-digital (inputs) and digital-to-analogue (outputs) conversions. Additionally the microprocessor

required data storage in order to perform the control algorithm. Data processing was also executed by the microprocessor (signal filtering, variables estimation, breaker position, alarms, etc.). The real-time synchronisation provided by the operating system was designed to run in a sequential basis [Donaghey, 1976]. Execution of control algorithms, data acquisition and data analysis in real-time were programmed in the FORTRAN language using 8-bit technology [East, 1984].

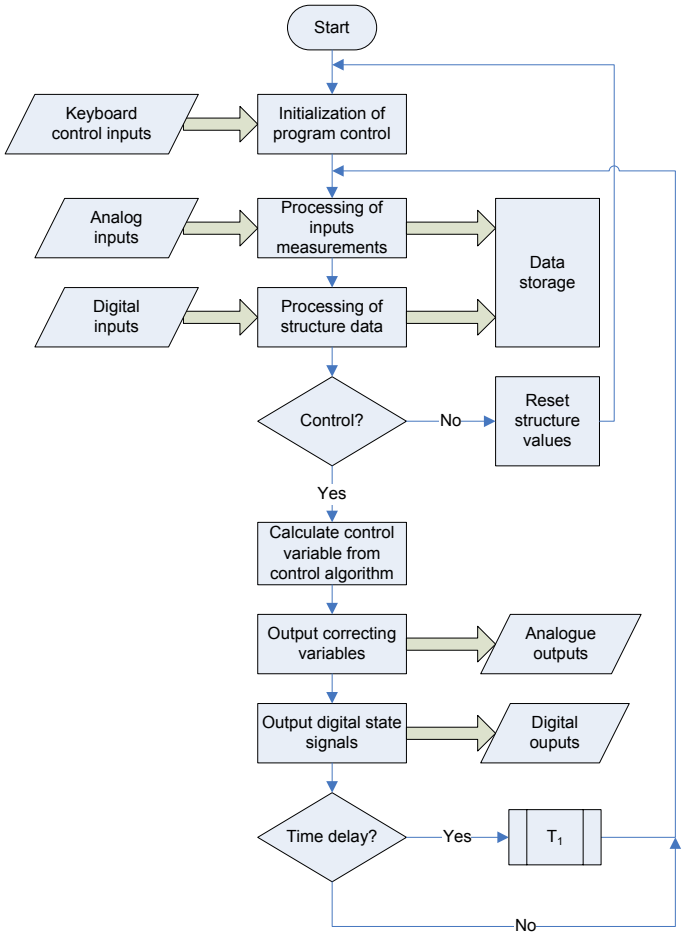

Figure 2-3 General process control program used in the second generation of microprocessors Flow diagram adapted from [Donaghey, 1976]

The core of the operating system –the scheduler– is responsible for assigning priorities in the execution of the control tasks [Donaghey, 1976]. In applications where computational resources were highly constrained, schedulers became

popular. In the early stage of control applications they were used with a fixed and predefined execution sequence for acquisition, control and actuation tasks [Butazzo, 2005].

PID control algorithms and other forms of control were implemented using either operating systems or even simple schedulers in customised applications. For instance, in the chemical industry cascade control (PI with inner-loop controller), nonlinear half-proportional, deadbeat response control, optimum state feedback, compensator control, adaptive control with linear filtering (3^{rd} order) were some of the common control algorithms using embedded systems [Donaghey, 1976].

Introduction of new control structures was primarily limited by the memory available [Donaghey, 1976]. On the other hand, the PID designs were relatively simple to program with the 8008, 6800 and Z80 technologies due to fewer requirements in memory and CPU load. Additionally such microprocessors supported floating-point mathematical operations through software subroutines [Maples, 1975].

As it was shown in the previous generation, many control applications in the 1970s using embedded systems substituted the common use of desktop computers. Nevertheless, for some chemical plant applications, the computing capacity of embedded systems was not able at that time to compete with analogue controllers [Donaghey, 1976]. The required computational resources for those applications were only found in desktop microprocessors. This might have been the reason that Intel hardware (e.g. 8008) evolved into general-purpose power microprocessors that eventually were confined to desktop applications. In contrast, Zilog (e.g. Z80) continued the path of embedded applications in almost all of its subsequent developments [Morse et al., 1980].

In the late 1970s, some control applications became economically feasible only when embedded processors were incorporated, for instance: missile guidance control [Mersten, 1979]; medical issues like the prosthetic arm described in [Graupe et al., 1978]; high precision machine tools [Lamb and Whelan, 1976]; control of solar heating systems [Eisenberg et al., 1977]. In some of theses cases, implementation of the control algorithm on embedded microprocessors allowed important advances with respect to previous technologies. For example, in [Graupe et al., 1978] it is shown that the implementation of an artificial limb without microprocessors was simply impossible.

Due to the flexibility offered by embedded systems, digital PID algorithms outperformed the analogue designs used in servo-systems [Nakamura et al., 2004]. This was particularly useful in multiple loop interactions such as cascade and feed forward [Auslander, 1996].

The software for multi-loop control systems was implemented with a main program and an interrupt service subroutine [Ahson et al., 1983]. This multi-loop approach used a combination of various program modules with different control algorithms, for instance: dead-beat, Kalman and Dahlin. In microprocessors such as the Intel 8085 each algorithm was accommodated in 1k byte of memory: see description in [Ahson et al., 1983]. The complexity of these algorithms in terms of computational resources was close to the discrete PID feedback controller. For instance, [Ellis, 2004], in "control systems design guide" shows how the lead/lag and several forms of PID control could be implemented. A discussion of the implementation on 8-bit microprocessors of such algorithms using a distributed arithmetic approach is shown in [Ahson et al., 1983].

The use of "larger" (in terms of memory) algorithms than PID started to be more common in the early 1980s. For example, nonlinear control algorithms

(integrated error squared with a variable dead-zone) reported important savings in comparison with the analogue controllers [Lipták, 2006].

2.2.3 Third generation of ECSs

For special control purposes, microprocessors evolved into microcontrollers [Schlett, 1998]. This technology integrated a CPU core with some basic peripherals (memory, I/O ports, timers, analogue-to-digital converters, pulse width modulation) in order to integrate a self contained device for more specific control purposes [Ganssle and Barr, 2003]. The more influential microprocessors were upgraded into microcontrollers; for instance, 8080 to 8048; 6800 to 6805, 6808, 6811 and Z80 to Z84.

In the 1980s, embedded technology was initially oriented to control of electric machines. See for instance [Rauta et al., 1996], [Wang, 1981] and [Ahson et al., 1983].

Digital microcontrollers replaced the conventional analogue servo controllers for different types of electric motors [Wang, 1981]. Some control techniques were too expensive or too complicated (interaction of multiple loops) to be implemented using analogue systems. By contrast, VLSI technology made possible the integration of different digital devices in one chip. The microcontroller structure allowed independent operation of the peripheral systems that improved both the performance of the system and the CPU overhead [Rauta et al., 1996]. These factors opened the door to cost-effective applications employing microcontrollers.

In motor-control applications over the 1980s, the US alone used 80 million motors in electronic printers, disk drives, tape drives and similar devices [Bose, 1988]. This tendency was followed in other motion applications, for instance industrial robots, numerical control machines and general-purpose industrial drives. Home

applications included appliance drivers for washing machines, dryers, air conditioners, blenders, mixers, etc. In order to cope with such requirements, position and velocity control as well as monitoring of system parameters were a vibrant research topic over that period [Wang, 1981].

As result of servomotor control research, new techniques were developed. For example, Blaschke introduced the direct or feedback method of vector control [Bose, 1988]. Similarly, Hasse proposed the indirect feed-forward control method [Bose, 1988]. Modern control algorithms were introduced for advanced military control applications [Gully and Coleman, 1981]. Linear quadratic (LQ) controllers and modern observer theory were widely exploited in state-of-the-art applications using embedded systems [Korn et al., 1981].

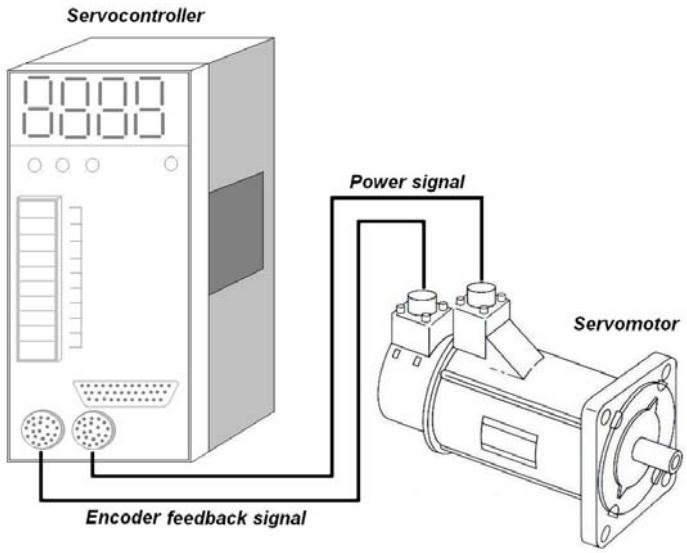

Figure 2-4 The third generation of embedded system was featured for vast contributions in the field of servo-controllers (adapted from [Dote, 1990] Figure 1.6).

A study presented in [Rajulu and Rajaraman, 1984] examined the suitability of microprocessors for control applications in relation to the execution-time using several algorithms. This study also compared the performance of several

microprocessors (16-bit and 32-bit) executing floating-point arithmetic operations. The fact of considering such computing elements for control algorithm implementation was merely inspired due to the inefficiency to find cost-effective solutions in some embedded control applications.

The computational capacity (memory and CPU load) used to implement Linear Quadratic Gaussian (LQG) Control in embedded systems was first presented in [Farrar and Eidens, 1980]. LQG methods have been tested in many aerospace applications where accurate models are characterised [Gu et al., 2005]. The relatively low computational load of such algorithms makes them suitable for embedded systems [Farrar and Eidens, 1980], however due to poor robustness for industrial applications [Gu et al., 2005], this approach has not been widely used in real-time applications.

The introduction of variable structure control algorithms in microprocessors started in robotics applications [Bose, 1988]. Robotics was –in fact– considered a major area of research in the late 1980s [Machado and De Carvalho, 1988]. Sliding mode control (SMC) was the pioneer of the non-linear algorithms implemented in this generation of embedded systems. The low computational complexity was an attractive feature of this kind of controller [Machado and De Carvalho, 1988].

Inadequate performance with the linear approaches and the high computational burden of adaptive control schemes made SMC a practical choice for high performance robot systems [Machado and De Carvalho, 1988].

2.2.4 Fourth generation of ECSs

The memory limitations and CPU performance of both microcontrollers and microprocessors used for advanced control and processing algorithms motivated further hardware developments in embedded systems [Dote, 1990]. One approach

invented in the 1960s and exploited almost 20 years later was the reduced instruction set computer (RISC), see [Bose, 1988]. This technology proposed an important simplification mnemonic instruction codes (compared with other hardware platforms). This was widely adopted in control applications with low-cost embedded microcontrollers [Leach, 1995].

The common use of compilers in this generation of embedded systems introduced additional challenges to the implementation of efficient code. Fast processors were developed for such purposes; for example, the digital signal processors (DSP), which were used to process very fast calculations [Dote, 1990]. Signal processing used in acquisition and control, made heavy use of DSPs and has been extensively documented [Embree, 1995]. Additionally, advanced processing algorithms (like observers) using DSP, were used in industry for sensorless applications [Dote, 1990]. The capacity of compilers to codify floating-point operations directly reduced the execution time of complex mathematical operations [Embree, 1995].

Diverse forms of PID, in single and multi-loop form, were still the predominant algorithm used with DSPs [Dote, 1990]. Nevertheless, the limitations of the PID design to address control problems due to non-linear plants with delays motivated the use of some more sophisticated control algorithms. One instance is the auto-tuning PID based on a neural network (N-N) [Dicheng et al., 2004] Moreover, adaptive control combined with PID and N-N is described in [Sheng et al., 2002] using the microcontroller 80C196KF3. These approaches were meant to combine the simplicity of PID and self-learning features of using adaptive N-N structures. An example of the industrial application of such techniques is the Foxboro M-760 controller [Bose, 1988].

Single and combined approaches based on both PID and fuzzy logic algorithms were developed for Motorola microcontrollers [Leach, 1995], [Jackson, 1997]. In

addition, a recent microcontroller MC68HC12 (16-bit core) included special hardware to perform fuzzy logic operations directly [Jackson, 1997].

Introduction of these advanced embedded systems added intelligence and diagnostic capacity to the control system [Bose, 1988].

In multivariable applications, the DSP was the first processor family suitable to implement sophisticated control algorithms [Dote, 1990]. DSPs achieved higher sample-frequencies than conventional 16-bit processors of the 1990s, see Table 2-1.

Due to their high cost, embedded systems based on DSPs were mainly employed in expensive applications. This motivated many contributions in the field of advanced computational software optimisation [Laub et al., 1987], [Balakrishnan et al., 2001], [Raghunath and Parhi, 1994]. Advance control, acquisition and signal processing were possible with the inexpensive technology of 8-bit and 16-bit microprocessors. For instance, the Z-80, 6502 and 68000 were used to implement adaptive control algorithms based on model reduction [Bruijn et al., 1988].

Table 2-1 Achievable sampling frequencies of different microprocessors (table taken from [Dote, 1990])

Microprocessor	*Clock*	*Sample frequency*
8086	8 MHz	<2 kHz
Z8000	5 MHz	<2 kHz
68000	10 MHz	<4 kHz
32016	10 MHz	<5 kHz
DSP TMS32010		31 KHz

Model reduction algorithms appeared as a functional tool in control design software; for instance, MATLAB or Simnon. In addition to this and with the

introduction of new and affordable families of 32-bit microprocessors and microcontrollers the use of more complex strategies for control applications was made possible.

Advanced RISC machinery (ARM) microcontroller technology incorporated multiple advantages compared to previous 32-bit microprocessors [Schlett, 1998]. Although cost-efficiency and low power consumption were the common benchmark with other 32-bit embedded processors (Motorola 68000), ARM introduced a new strategy to reduce the code density. This strategy was based on a subset of 32-bit ARM instructions recoded into 16-bit opcodes.

Several control application using ARM embedded processors have been reported, in both research and industrial applications. For instance, [Chen et al., 2007] presented a highly complex distributed embedded controller with multiple forms of control. In such application, an ARM microcontroller is used as an interface to a personal computer (dedicated, for example, to control planning trajectories using neural-network algorithms). USB[3] is employed to communicate commands from the computer to ARM microcontroller. This system incorporates a program to translate the high-level PC commands into low-level commands for the specific purpose controllers that implement PID control. The internal system's communication is based on the CAN[4] bus.

The theoretical contribution of optimal FLC+PID designs [Tang et al., 2001], [Pavlica and Petrovacki, 1998], [Huang and Yasunobu, 2000] inspired some real-time applications. For instance, motor control [Betin et al., 2000] and pressure control processes (using a hybrid algorithm FLC+PI) [Kanagaraj et al., 2008]. The use of such algorithms became common practice for ARM processors. In general, these algorithms are based on a small set of linguistic rules and their

[3] Universal serial bus
[4] Controller area network

implementation does not require big memory allocation. The mechanism of FLC avoids modern control algorithms along with complex model representations [Passino and Yurkovich, 1998]. However, for relatively complex systems with MIMO structures, selection of an appropriate set of linguistic rules has led to auto-learning approaches [Jang, 1993], [Nomura et al., 1996], [Zhou et al., 2003], [Passino and Yurkovich, 1998]. Consequently, the complexity of the algorithm increases and implementation of this algorithm for low-cost processors depends strongly on the memory and CPU performance available.

In the hardware terrain, the new technology that emerged in the evolution of more sophisticated hardware for control was the field programmable gate array (FPGA). FPGAs are the result of development in the field of the programmable gate array (PLD) and complex PLDs (CPLD). Today's FPGAs can contain up to 2 PowerPC processors, 10 MegaBytes of memory, and 11.1 GigaBytes/s IO, along with other logic gate enhanced features [Sevcik, 2006].

Multiple efforts between companies and universities have been made in order to adopt this relatively new technology in embedded systems designs [Sevcik, 2006].

The FPGA provides fast digital implementation of control systems, for instance in [Shouling and Xuping, 2007] an embedded hardware/software co-design of a hybrid control algorithm is performed. An adaptive PID estimates the parameters online, in contrast to the traditional offline implementations. The computational capacity of the FPGA for this application allows execution of parameter identification using an ADALINE neural network. Although experimental results indicate satisfactory performance, the cost of the overall system is higher than with the stand-alone microcontroller used in similar applications.

A relative new control technique currently used for nonlinear and multivariable systems is the model predictive control (MPC). MPC has been successfully tested

in the petrochemical industry, and is currently used in a wide range of industrial processes [Ling et al., 2006].

In embedded systems, the design of an MPC algorithm depends fundamentally on the application. In general, this algorithm requires high computational capacity. However, the use of FPGAs offers important advantages for the efficient implementation of floating-point operations and matrix inversions, since these devices contain specialised modules that perform such operations directly. For instance, the Spartan-3L FPGA with 1.5 million gates can handle a 128x128 matrix inversion problem with IEEE single precision floating point arithmetic (8-bit exponent and 23-bit mantissa) [Ling et al., 2006].

Other applications of MPC in embedded systems can be found in micro-chemical and drug-delivery systems [Parker et al., 1999].

The parallel execution of control algorithms using FPGAs allows exploration of more complex control designs. For instance, a LQ design is reported in [Piotr-Jastrzebski, 2007] used for high precision control of active magnetic bearings.

Today, different software is available to design control systems using FPGAs. Nevertheless, the compilation time is still growing faster than the computational power (it may take hours or even days to complete a compilation of a 2-million gate chip [Fobel et al., 2007]).

In the PLC domain, the fourth generation of embedded systems has reached tremendous functionality in terms of software support. Five standard languages have been established for these industrial controllers: ladder diagram (LD); function block diagrams (FBD); structured text (ST); instruction list (IL) and sequential function chart (SFC) [Colnaric, 1999].

The extensive use of LD has been substituted for highly functional software for SFC. It has improved tremendously the design and maintenance of code in this matter. However, the practice of automatic control in PLC remains the standard choice for use in simple controllers like PID. A few exceptions can be found in some PLC vendors (e.g. Omron), that include some advanced control algorithms, such as FLC.

Next generations of embedded systems hardware indicate a clear tendency towards wide use of reconfigurable hardware (such as FPGAs). For control applications, this opens new possibilities to use advanced control algorithms (in comparison to the traditional PID) [Ling et al., 2006], [Shouling and Xuping, 2007]. The way to customise this reconfigurable hardware is still an open problem.

Additionally, multi-core processor technology is a good alternative to develop efficient, single and multipoint distributed embedded applications. In order to achieve high performance and power reduction, the ideal hardware platform is still a matter of debate. In this context, homogeneous and heterogeneous multi-core processor architectures have been recently under intensive research according to [Pozzi and Paulin, 2007].

2.3 Do ECSs have enough support?

After nearly 40 years of evolution, embedded systems is considered one of the most important sectors for semiconductor applications [WSTS, 2005]. Based on this fact, a brief survey is presented next; it explores what supporting tools (theoretical and practical) have been developed to facilitate implementation for ECSs.

2.3.1 Which algorithms have been supported for ECSs?

As it was shown in the survey presented in the Section 2.2 PID algorithms (P, PD, PI, lead-lag) are the most dominant forms of automatic control in the context of embedded systems. Additionally, in practice, there are multiple techniques in order to deal with their tuning procedures using digital computers. For example, the well known tuning method Ziegler-Nichols [Franklin et al., 1993] widely used for proportional, proportional integral and PID controllers in different applications. Anti-windup algorithms prevent actuator saturation, thus avoiding overshoots and plant instability [Ellis, 2004]. Some methods were designed to avoid problems associated with the derivative action, even when a filter is used [Atherton and Majhi, 1999]. Additionally, "bumpless" action technique was implemented to avoid an aggressive jump of the output signal caused by the commutation between the manual and automatic control stages [Youbin et al., 1996]. Design of feed-forward strategies combined with the classic PID controller were used to deal with plant parameters uncertainties [Jones et al., 1996].

Most of these techniques and many more have successfully improved the general performance of the systems. However, some applications using PID in high performance mechatronic systems still face multiple challenges. For instance, in [Lin, 1989], dry friction and gravity vector forces cannot be compensated using this approach. In contrast to some other applications for example [Arteaga and Kelly, 2004], the implementation of PID provides high performance. However, it requires expensive actuators in order to give the desired performance. In theoretical decoupling approaches such as [Coelho et al., 2004] there are problems in the algorithm's implementation for real industrial applications. In general, the payload is not fixed and the daily work increases system uncertainties. Similarly, for hybrid approaches (like PID+Neural-Networks approaches or FLC) which still require multiple support tools to make some applications cost-effective: see for instance, [Lewis, 2005] and [Sun and Meng, 2004]. The effectiveness of PID for

some applications can open the discussion on incorporation other of, non-traditional, forms of control in order to improve performance.

In this context, embedded software to support control application in embedded systems plays an important role in the efficient implementation in current and future hardware.

2.3.2 Is the software ready to catch up with hardware?

Some reasons that cause a mismatch between hardware and software development in embedded systems are described below:

1) In recent years, one of the most important challenges is related to improving the power efficiency for microprocessors (MIPS/watt ratio). An extensive survey in this topic is presented in [Venkatachalam and Franz, 2005]. Instead of software, hardware improvements have played the central role in this matter [Moyer, 2001].
2) The rapid demand of high performance devices is currently the driving force to develop embedded systems with multiple functionalities [Wolf et al., 2002].
3) There is no efficient software to deal with high code density [Leupers, 2002]. Instead, the main contributions are based on hardware.
4) Very-large-scale integration (VLSI) helped to introduce processing capabilities. The advances in multimedia functions have been based on the potential power given by the VLSI revolution [Dahlgren, 2001]. Emerging hardware developments force the software development to handle multiple functionalities. In order to meet the increase in hardware capacity, software development has to innovate at the same level. This becomes more complicated when there is a tendency to reduce the product's life cycle.
5) Embedded hardware is traditionally "disposable" technology; new components substitute previous devices. In contrast, embedded software demands tools to deal with maintenance issues, which add to the cost of the

final product. For instance, for the US department of defence the software maintenance cost for some mission critical applications represented $110 per line of code, while the cost for non-embedded software was $5.60 per line of code [Clark et al., 1999].

In the end, customised software in embedded systems for specific applications has been constrained by the imbalance between hardware and software tools available.

Some research in embedded software has focused on the practices and tools used in order to increase the price/performance ratio of the products. For instance, embedded software for avionics development is one of the pioneering sectors that have introduced new tools in the integration (software into emerging hardware) [Audsley et al., 2003]. In 2003, software/hardware verification and validation for this commercial avionic application represented between 40-50 percent of the total software integration budget [Prisaznuk, 2003]. The reason that motivates research and development of efficient integration was not only the cost but also safety issues [Storey, 1996].

In other products such as mobile phones, commercial pressure has led to the increase of multi-functionality. Consequently, code size has increased enormously. It is estimated that by 2010 a typical mobile phone will have 20 million lines of embedded software [Charette, 2005]. By the same year, every car produced for General Motors Co. will require 100 million lines of code [Charette, 2005]. The consequences of adding complexity to the software often constrains integration with hardware, and increases the cost.

Previous research to various strategies to facilitate integration of embedded software is shown in [Napper, 1998]. According to this research, there are four reasons that affect integration. The lack of good automation tools (39 percent)

and shortage of trained engineering resources (38 percent), design teams require more software expertise (15 percent) and tools are becoming as complex as chips (8 percent).

This study revealed that 47 percent of developers were interested in code reuse and 27 percent in hardware/software co-design tools in order to deal with the integration problem.

A recent study has reported a similar tendency to that, see [Napper, 1998]. It shows how the gap between the software tools and practical implementation affects developers' productivity [Dormoy, 2005]. In the same context, some leading European companies are becoming cautious regarding the inclusion of new technology due to difficulties at the integration stage [Graaf et al., 2002].

The following sub-section surveys the science involved in integrating software and hardware for low-cost embedded systems.

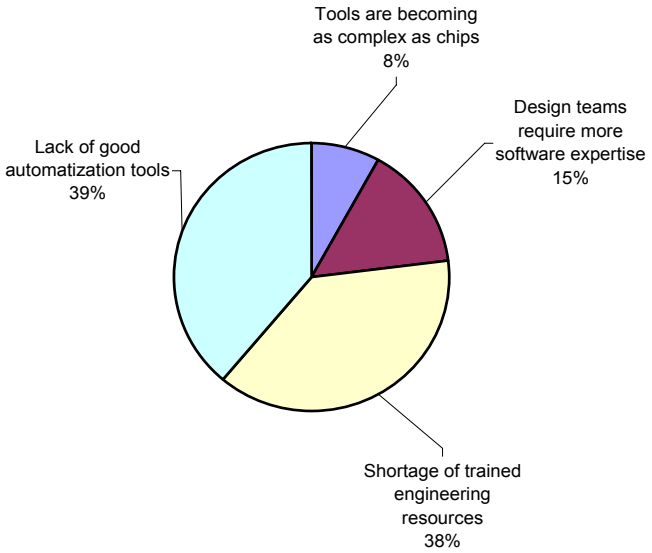

Figure 2-5 Factors that affect embedded system design integration between the new hardware and software (Aisys Inc. [Napper, 1998])

2.4 Supporting tools for embedded systems applications

For economic reasons, industry has set up initiatives to standardise the software architecture in order to facilitate complex implementation scenarios [Dormoy, 2005], [Reynaert, 2008]. Nowadays these software tools are ubiquitous for some applications [Teng, 2000], [Martin et al., 2001], [Maclay, 1997].

The evolution and impact of the most relevant approaches in the field of hardware-software integration is presented next.

Perhaps the most influential software tools used to facilitate integration of the software in hardware are the program compilers. In general, these programs are able to link high-level language into machine code [Sammet, 1969]. The history of compilers dates back to 1954 with the development of FORTRAN I compiler [Backus, 1978]. The IBM team led by John W. Backus designed this compiler to reduce the job of coding and debugging. The continuation of the compiler evolution is detailed in "History of Programming Languages" [Bergin and Gibson, 1996].

Although these tools emerged in non-embedded applications, they have been adopted for embedded systems. In general, mid-level languages showed important advantages in terms of the efficiency of the written code. In fact, nowadays the most common language for embedded systems is "C" [Schlett, 1998], [Pont and Banner, 2002].

While the C programming language translates either mid-level or high-level programmes into object-code [Bergin and Gibson, 1996] new tools have been used to further simplify the codification problem. For instance, the unified modelling language (UML) is a tool designed to specify, visualise, construct and document software for general purpose systems [Lange et al., 2006]. UML shows important advantages for desktop applications. Several efforts are being made in

order to design a well defined modelling language for embedded systems [Kukkala et al., 2005], [Vanderperren and Dehaene, 2005]. Presently there is a mismatch between UML design models and embedded software designs due to timing constrains, limited memory and CPU power [Schatz et al., 2003], [Martin et al., 2001].

A recent extension of UML is the system-modelling language (sysML) [SysML, 2005]. This approach is a customised UML for systems engineering applications that can be used to integrate hardware and software. However, for embedded systems this tool still lacks formalisation [Mwelwa, 2006].

Similar software tools to UML deal also with specific engineering requirements in order to cope with the code-automation problem. For instance, integration methods like model-based code generation [Schulz et al., 1998], [Hammarström and Nilsson, 2006], [Nossal and Lang, 2002] are used for:

1) Design application with desired requirements rather than random specifications.
2) Communicate design ideas across the design team.
3) Provide maintainability in the design.

Some advanced tools, based on auto-code generation, deal with the complexity of acquisition and control of real implementations. For instance, LabView RT [Mrad et al., 2000]; real-time MATLAB toolbox [Teng, 2000]; simulink/RTW and hard real-time Linux environment [Quaranta and Mantegazza, 2001]; FWR MATLAB toolbox (INRIA); TrueTime (University of Lund) [Henriksson, 2006] and MATLAB RTW Toolbox MIRCOS 167 [Wörnle and Murillo-Garcia, 2002]. These technologies offer to speed-up the implementation of computational designs.

In connection with all these software tools, dedicated hardware was developed in order to provide an integral experimental toolset. This dedicated hardware has been used to simplify the design-implementation loop. Two commercial technologies like hardware in the loop [Maclay, 1997] and rapid prototyping [Wootaik et al., 2004] have been extended recently in the automotive sector, in particular, for safety critical applications [Nossal and Lang, 2002]. However, software integration for low-cost applications has limited support using these tools [Wörnle and Murillo-Garcia, 2002].

Limited support for low-cost embedded systems is related to:

1) Both limited memory and CPU performance that often constrain implementations of relatively large and complex systems. It is unlikely to increase the flexibility and adaptability of the system for different circumstances [Henriksson, 2006].
2) Software tools have not been developed for some microcontroller families, and even when they are supported, it requires specific knowledge of the target hardware in order to use these tools efficiently [Nair et al., 2004], [Hammarström and Nilsson, 2006].
3) There is a trade-off between the complexity of obtaining an accurate plant model and the minimum performance requirements [Åström, 1985].

2.5 Conclusions

In this chapter, the history of embedded system hardware and the control algorithms used for these platforms was reviewed.

Among the control algorithms used for embedded systems, Fuzzy Logic Control (FLC) and hybrid approaches have shown satisfactory performance for different applications. Therefore, it is worth considering a study to find practical methods to support their implementation in low-cost embedded processors.

The open problem to implement FLC in embedded processor is related to the complexity of reducing the rules efficiently without affecting the performance and robustness of the system.

Similarly, other forms of control like Linear Quadratic Regulator (LQR) and H-infinity have not been efficiently supported for low cost platforms. These techniques can be integrated in common industrial practice where traditional techniques cannot achieve the performance or robustness desired.

The latest technology in embedded systems came to reduce the cost and increase the performance. However, the steep learning curves that accompany both product development and modern control algorithms reduce the incorporation of such technology in many industrial applications.

Some issues related to the optimisation of embedded systems were explored. It was shown that the principal impediment to reduce the cost in embedded systems is the clear imbalance between the hardware and software development. The support in many embedded control applications is becoming costly and ineffective. In this context, the creation of effective methods to facilitate integration between control theory, software and hardware is required.

Chapter 3
The testbed

This chapter provides the description of the testbed used in this thesis. A model of the plant along with the specifications of the control hardware is included.

Information related to the benchmark controller is introduced including additional implementation and relevant issues of the testbed.

3.1 Introduction

Mechanical, electronic and control software for four different testbeds were designed for this research. These mechatronic systems were: 1) a robot with three degrees of freedom, 2) a micro-mobile robot (two driven wheels), 3) a servo system based on a brushless DC motor and 4) a sub-actuated robot (inverted pendulum). The design of these systems enriched the "know-how" of the whole project. However, the only apparatus employed as a case study for published papers was the inverted pendulum. The relative difficulty (non-linear system SIMO[5] system) of the inverted pendulum and the inherent instability in open loop suggested its appropriateness.

3.2 General description of the testbed

Dynamic systems are described using differential equations; conventional control algorithms are designed to maintain these systems stable. However, in low-cost applications the system model must to include uncertainties in actuators, sensors and controllers. This is due to the nonlinearities of the components that cause a

[5] SIMO stands for "Single Input Multiple Output".

mismatch between the ideal and actual model. When this mismatch is largely inconsistent, it is almost impossible to design a proper controller by using traditional approaches. In order to provide an accurate model of the testbed, inclusion of the actuator dynamics was necessary. Chapter 7 includes a dynamic model of the sensors of this testbed.

The inverted pendulum has two equilibrium points, one is stable (down position) and the other is instable (upright position). The unstable equilibrium point is the focus of the analysis presented in the next section.

Figure 3-1 shows the basic free body representation of the inverted pendulum.

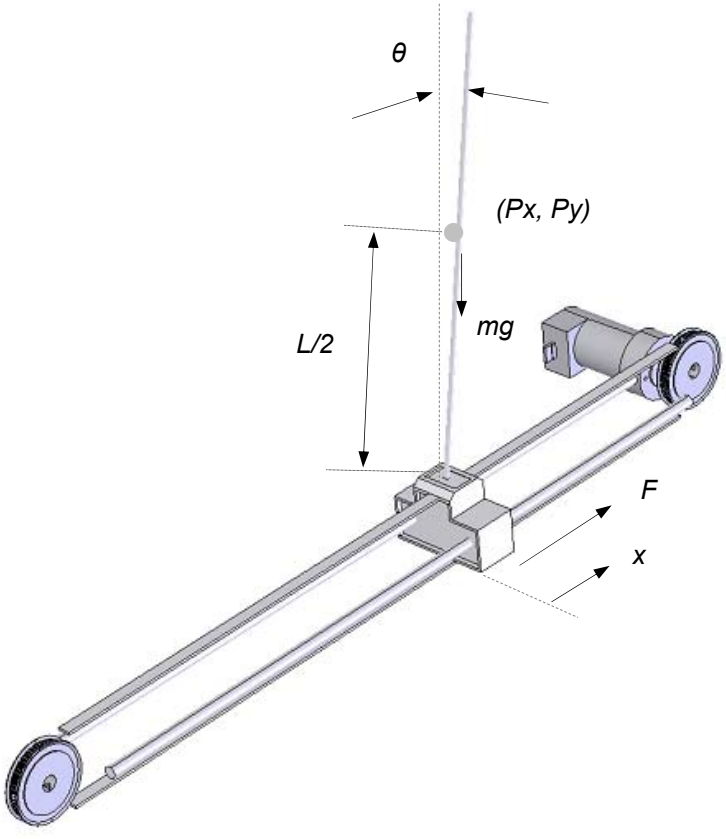

Figure 3-1 Diagram of the inverted pendulum and its variables (figure taken from [Bautista-Quintero et al., 2005])

Note that the control goal is not only to hold the pendulum upright, but it also to maintain the cart at a pre-defined "set point" (SP) position on the track (Figure 3-1). As a result, it is essential to measure both the rod angle and the cart position. In both cases, incremental encoder sensors are attached to the system to achieve this.

Figure 3-2 shows the motor assembly used to drive the pendulum, along with the gearbox and integrated encoder.

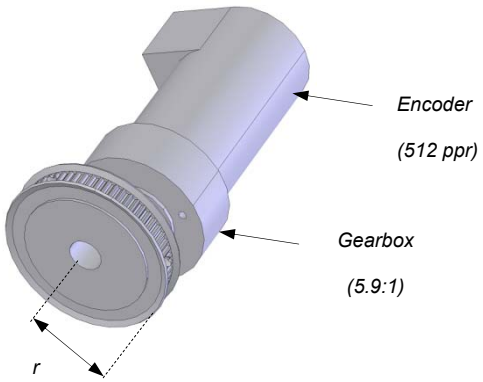

Figure 3-2: The pendulum is driven by a DC motor with integrated gearbox and encoder, figure taken from [Bautista-Quintero et al., 2005].

Equation 3-1 defines the relation between the pulses given by the encoder and the position of the pendulum cart:

$$\beta_1 = \frac{2\pi \cdot r}{\mathrm{Re}s\_Enc_{CAR} \cdot Gear\_box} \qquad \text{Equation 3-1}$$

where r is the radius of the pulley, *Res_Enc* is the resolution of the encoder and *Gear_Box* is the reduction ratio of the gears in the motor.

For the rotational sensor in the base of the rod, another encoder is attached. The resolution is given in Equation 3-2:

$$\beta_2 = \frac{2\pi}{\mathrm{Re}s\_Enc_{ROD}} \qquad \text{Equation 3-2}$$

Table 3-1 Parameters of the sensors and actuator

Constant parameters	*Abbreviation*	*Value*	*Units*
Pulley radius	*r*	0.036	metres
Resolution of the encoder_c	Res_Enc_{CAR}	512	ppr
Resolution of the encoder_r	Res_Enc_{ROD}	4000[6]	Ppr
Gear box ratio	*Gear_box*	5.9	Non-dimensional

An ARM microcontroller is used to drive the motor. It uses a 10-bit resolution PWM signal that is adapted (conditioner circuit) and amplified (power amplifier) in current and voltage in order to provide the control signal to the actuator. The relation between the digital signal u and the real voltage applied to the system is directly proportional to the constant β_0, see Figure 3-3.

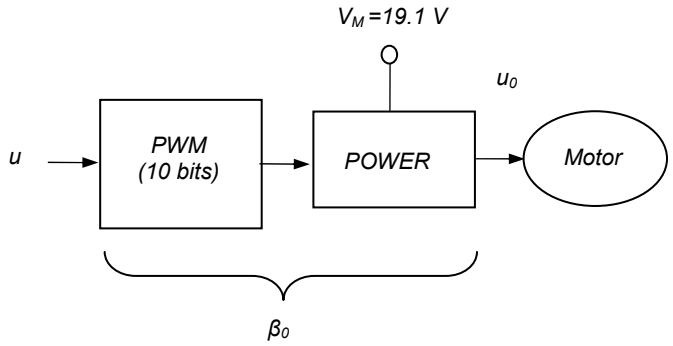

Figure 3-3: The interface between the PWM output from the microcontroller and the motor itself. For this system $\beta_0 = V_M/2^{Bits\_PWM}$.

Based on appropriate empirical tests, the actual parameters are listed in Table 3-2

3.2.1 Dynamic model of the testbed

Using the Euler-Lagrange representation [Spong and Vidyasagar, 1989] it is possible to obtain a nonlinear mathematical model of the testbed: such an

[6] Some of the experiments in this thesis will use encoders of 2000 and 1000 ppr

approach considers the kinetic and potential energy of the mechanical chain and obtains a model of the torque at each link.

Table 3-2: Real motor and sensor parameters

Parameter	***Description***
β_0=19.1/1024 Bits/Volt	Dimension factor of the voltage (Bits_PWM =10 bits)
β_1=6.8327e-5 Metres/Pulses	Dimension factor of the linear position (related to encoder, pulley and gearbox)
β_2=1.57e-3 Radian/pulses	Dimension factor of the angular position (related to the rod sensor)

The centre of gravity of the rod is located at *Px, Py,* where:

$$p_x = x + l\sin\theta$$
$$P_y = l\cos\theta \qquad \text{Equation 3-3}$$

The moment of inertia of the rod is J, which is approximately:

$$J = \frac{1}{12}(mL^2) \qquad \text{Equation 3-4}$$

The kinetic energy of the pendulum is given by the following equation,

$$\frac{m}{2}(\dot{p}_x^2 + \dot{p}_y^2) + \frac{J}{2}\dot{\theta}^2 \qquad \text{Equation 3-5}$$

where the rod's mass *is m*.

The total kinetic energy is given in Equation 3-6,

$$K = \frac{M_c}{2}\dot{x}^2 + \frac{m}{2}\left[(\dot{x} + l\dot{\theta}\cos\theta)^2 + (l\dot{\theta}\sin\theta)^2\right] + \frac{J}{2}\dot{\theta}^2 \qquad \text{Equation 3-6}$$

where *l* is *L/2* and *Mc* is the mass of the cart.

The Rayleigh dissipation function is:

$$D = \frac{1}{2}\mu_x \dot{x}^2 + \frac{1}{2}\mu_\theta \dot{\theta}^2 \qquad \text{Equation 3-7}$$

The potential energy is:

$$P = mgl\cos\theta \qquad \text{Equation 3-8}$$

In order to represent the general movement of the system we substitute the above results in the Lagrange equation:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = \tau_i \qquad \text{Equation 3-9}$$

The general coordinates are: $q_1 = x, \quad q_2 = \theta, \quad \tau_1 = F, \quad \tau_2 = 0$

Substituting Equation 3-6, Equation 3-7 and Equation 3-8 in Equation 3-9, Equation 3-10 is obtained. This nonlinear expression represents the mathematical model of the pendulum:

$$\begin{aligned} (M+m)\ddot{x} + ml\cos\theta\ddot{\theta} - ml\dot{\theta}^2\sin\theta + \mu_x\dot{x} &= F \\ ml\cos\theta\ddot{x} + (J+ml^2)\ddot{\theta} - mgl\sin\theta + \mu_\theta\ddot{\theta} &= 0 \end{aligned} \qquad \text{Equation 3-10}$$

In order to simplify the model around the unstable equilibrium point ($\theta=0$) we have: $\sin\theta \approx \theta,\ \cos\theta \approx 1,\ \dot{\theta}^2\sin\theta \approx 0,\ \mu_x\dot{x} \approx 0, \mu_\theta\dot{\theta} = 0$

Thereby the model given in Equation 3-10 can be approximated by Equation 3-11:

$$\begin{aligned} (M+m)\ddot{x} + ml\ddot{\theta} &= F \\ ml\ddot{x} + (J+ml^2)\ddot{\theta} - mgl\theta &= 0 \end{aligned} \qquad \text{Equation 3-11}$$

As previously noted, a DC motor activates the cart. The model considers force as an input variable; therefore, it is crucial to find the dynamic relation between the input voltage u_0 and force F:

$$F = -\frac{J_M}{r^2}\ddot{x} - \frac{\alpha_2}{r}\dot{x} + \frac{\alpha_1}{r}u_0 = -M_o\ddot{x} - c_2\dot{x} + c_1u_0 \qquad \text{Equation 3-12}$$

where:

J_M	is the inertia of the rotor of the DC motor
r	is the radius of the pulley
α_1 and α_2	are factors that depend on the electric characteristics of the DC motor

Substituting Equation 3-12 in Equation 3-11, Equation 3-13 is obtained:

$$(M+m)\ddot{x} + ml\ddot{\theta} + c_2\dot{x} = c_1 u_0$$
$$ml\ddot{x} + (J+ml^2)\ddot{\theta} - mgl\theta = 0 \qquad \text{Equation 3-13}$$

Equation 3-13 shows the differential equation of the pendulum. In this, the states x and θ are variables that are expressed in metres and radians respectively.

For space-state representation, the following vector state is chosen:

$$\chi = \begin{bmatrix} x \\ \theta \\ \dot{x} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \qquad \text{Equation 3-14}$$

Simplifying, by using state equations:

$$\frac{d}{dt}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & -f_1 & -f_2 & 0 \\ 0 & f_3 & f_4 & 0 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ g_1 \\ -g_2 \end{bmatrix} u \qquad \text{Equation 3-15}$$

where:

$f_1 = m^2 l^2 g / H$
$f_2 = (J+ml^2)c_2 / H$
$f_3 = (M+m)mgl / H$
$f_4 = mlc_2 / H$
$g_1 = (J+ml^2)c_1 / H$
$g_2 = mlc_1 / H$

$$H = (M + m)J + Mml^2$$

The output vector then is:

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} \qquad \text{Equation 3-16}$$

3.2.2 Model parameterisation

Although most of the parameters can be characterised (i.e. obtain the closest to the real physical value) by direct measurement, for example, mass of the rod, the coefficients of the motor depend on the load (mass of the cart and friction in the rail), so that, these parameters require a different method in order to characterise them. A method to obtain such values is proposed next.

1. Obtain a partial dynamic model of the system; for this testbed this is the relation between the actuator and the linear movement of the cart.
2. Obtain the *Laplace* expression of this equation.
3. Find the solution in the time domain.
4. Obtain the parameters c_1 and c_2 based on measurements in the step input response and mass of the cart (M), see Figure 3-4.

The differential equation of the cart movement is given by Equation 3-17.

$$\frac{M}{c_2}\dot{v} + v = \frac{c_1}{c_2}u_0 \qquad \text{Equation 3-17}$$

Using the *Laplace* transform of Equation 3-17, is obtained in Equation 3-18 is obtained:

$$V(s) = \frac{\frac{c_2}{c_1}}{\frac{M}{c_2}s + 1} \qquad \text{Equation 3-18}$$

The following graph shows the results obtained by finding the solution to the step input and transforming to the time domain:

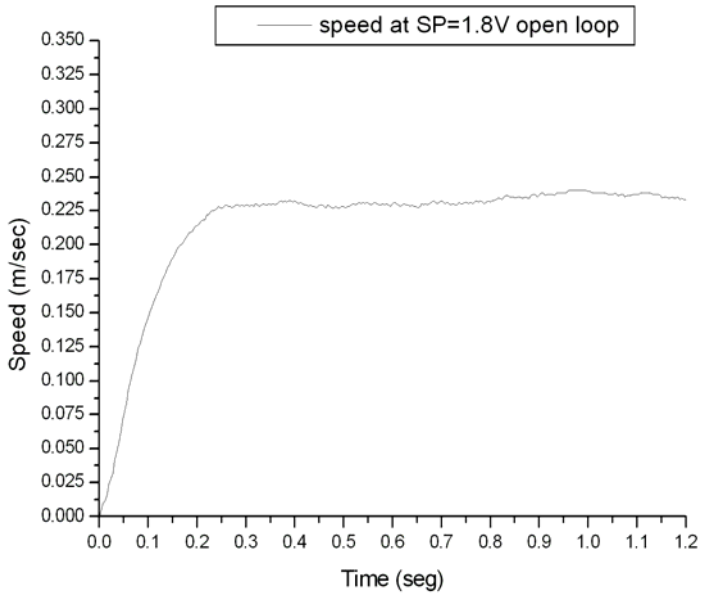

Figure 3-4. Open-loop response to the cart movement for a step input of 1.8 volts step input

$$v(t) = \frac{c_1}{c_2}(1 - e^{-\frac{t}{\tau}})u_0, \qquad \tau = \frac{M}{c_2}$$

Equation 3-19

Based on Equation 3-19 τ is obtained (time when 63.1% of the steady state value is reached). In addition, using Equation 3-19 the values of c_1 and c_2 can be calculated in the relation c_1/c_2 (i.e. voltage-velocity 1.8Volts/0.225 m/s). The parameters characterised are shown in Table 3-3.

3.3 The benchmark control implementation

Throughout this thesis, PID is used as the common benchmark algorithm for comparison with other forms of control. This benchmark control implementation

is described in this section. Description of the implementation is divided into three parts: data acquisition, control algorithm and actuation.

Table 3-3 Real parameters of the testbed

Constant	*Symbol*	*Value*	*Units*
Cart mass	*M*	0.028	kilograms
Rod mass	*m*	0.05	kilograms
Rod length	*l*	0.305	metres
Rod inertia	*j*	0.00155	kg metres^2
Gravity acceleration	*g*	9.81	m/s^2
Coefficient 1	c_1	4.48	Nw/Volts
Coefficient 2	c_2	38.14	kg metres/s

3.3.1 Data acquisition

The data acquisition system is designed to measure the rod and cart positions. Both sensors attached to these links are incremental optical encoders. For the measurements of the rod position (encoder shaft directly attached to the rod, see Figure 3-1), the sensor used is RI 32-0/1000ER, (1000 ppr). Moreover, for the measurements of the cart position (the encoder is indirectly attached to the cart movement using a gearbox and a pulley Figure 3-1).

A digital filter is implemented in order to eliminate the noise produced during the speed measurement process. A recursive filter is used for this application (see Equation 3-20).

$$\tilde{v}_f = \alpha\tilde{v}_f + (1-\alpha)\tilde{v} \qquad \text{Equation 3-20}$$

where:

$\tilde{v}_f$ Speed after the filter process

α Filter coefficient

This filter is based on obtaining the average of a signal.

$$\bar{x}_k = \frac{1}{n}\sum_{i=k-n+1}^{k} x_i \qquad \text{Equation 3-21}$$

Similarly, at the previous time instant, *k-1*, the average of the latest *n* samples is:

$$\bar{x}_{k-1} = \frac{1}{n}\sum_{i=k-n}^{k-1} x_i \qquad \text{Equation 3-22}$$

Therefore,

$$\bar{x}_k - \bar{x}_{k-1} = \frac{1}{n}\left[\sum_{i=k-n+1}^{k} x_i - \sum_{i=k-n}^{k-1} x_i\right] = \frac{1}{n}\left[x_k - x_{k-n}\right] \qquad \text{Equation 3-23}$$

This, after rearrangement:

$$\bar{x}_k = \bar{x}_{k-1} + \frac{1}{n}\left[x_k - x_{k-n}\right] \qquad \text{Equation 3-24}$$

If $\alpha = 1/n$:

$$\bar{x} = \alpha x_k + (1-\alpha)x_{k-1} \qquad \text{Equation 3-25}$$

The weight of α determines how smooth and fast the response of the filter will be. The bigger the value (near 1) the filter makes the change between the last sample and the current sample smoother. For smaller values (near 0) the filter has a faster response between the last sample and the current sample.

Such a filter is easy to implement even using resource constrained embedded processors.

3.3.2 Control algorithm

The control algorithm is the classic PID; its implementation is as follows.
The PID is made up of three parts, 1) action proportional to the error, 2) action proportional to the integral of the error and 3) action proportional to the derivative of the error. Equation 3-26 describes the PID action:

$$U(t) = K\left[e(t) + \frac{1}{T_I}\int e(\tau)\tau + T_D \frac{de(t)}{dt} \right] \qquad \text{Equation 3-26}$$

In order to obtain the integral part, several methods have been suggested for implementation in digital computers, see [Ellis, 2004]. Here, a simple -yet effective- integration method based on Euler's integration is proposed. This approximation is shown in Equation 3-27.

$$I_n = I_{n-1} + T_s R_n \qquad \text{Equation 3-27}$$

where: R is the input, I is the integrator output, T_s is the sample time.

Equation 3-27 can be translated to the z-domain to facilitate the algorithm's implementation:

$$I(z) = I(z)z^{-1} + T_s R(z) \qquad \text{Equation 3-28}$$

Likewise, the differential part is obtained based on the approximation of the inverse Euler's integration.

$$D_n = \frac{R_n - R_{n-1}}{T_s} \qquad \text{Equation 3-29}$$

Translating Equation 3-29 into the z-domain gives Equation 3-30

$$D(z) = \frac{R(z) - R(z)z^{-1}}{T_s} \qquad \text{Equation 3-30}$$

The pendulum requires two control regulators, a PID to control the cart and a PD to control the rod. The constants are, K_{Pcart}, K_{Icart}, K_{Dcart}, K_{Prod} *and* K_{Drod}.

The tuning process for the pendulum was based on trial-and-error using the following rules:

1. Change K_{Prod} from 0 to a certain value, such that the rod remains stable in the upright position regardless of the cart's position (the cart may move beyond the rail limits, in which case the experiment must be repeated). Keep the value of K_{Prod} and record the speed of the cart ($speed_{cart\_openloop}$).

2. Gradually increase K_{Drod} without causing instability to the pendulum (likewise in the previous step regardless of the cart's position). Keep the value.
3. Put K_{Prod}=0 and K_{Drod}=0, let the set point of the cart's position equal zero SP_{cart}=0, and adjust K_{Pcart} in such way that the cart moves away from the centre of the rail (assuming SP=0). Increase it until the cart has the same speed as in step 1 ($speed_{cart\_openloop}$). Keep the value.
4. Increase the gain K_{Drod} before the system becomes unstable. Keep the value.
5. Apply all gains together and test the pendulum.
6. If the system is unstable reduce K_{Pcart} and K_{Dcart}
7. Adjust K_{Icart} if there is a steady-state-error in the cart's position (this action will require an anti-windup reset).

The gains obtained for this testbed from the above procedure are: *K_{Pcart}=-4.18 volts/metres, K_{Icart}=1.5 volts/metres seconds K_{Dcart}=15.28 volts seconds/metres and K_{Prod}=-31.38 volts/rad and K_{Drod}=7.574 volts seconds/radians.*

3.3.3 Actuation section

The system has only one actuator, a DC servomotor. There are some considerations related to this motor in order to implement the code. For this system, there are three aspects to consider: 1) anti-windup algorithm, 2) dead zone in the actuator response, 3) sign change of the output using an external signal (in order to control the direction of motion in the motor).

3.4 Experimental results of the PID implementation

The discrete version of Equation 3-15 (using Tustin's method) of the model presented in Equation 3-15 is shown in Equation 3-31.

$$x(k+1)=\begin{bmatrix}1 & 0 & 0.0056 & 0\\ 0 & 1.0013 & 0.0109 & 0.01\\ 0 & -0.0070 & 0.2756 & 0\\ 0 & 0.2585 & 1.7927 & 1.0013\end{bmatrix}x(k)+\begin{bmatrix}0.005\\ -0.0013\\ 0.0856\\ -0.2106\end{bmatrix}u(k) \qquad \text{Equation 3-31}$$

T_s is the sample interval; like any control system (in a resource-constrained design), the selection of the sample interval involves a trade-off between performance and computational resources. For example, the shorter the values of Ts, the harder it is to meet the deadlines to execute acquisition, control algorithm and actuation tasks. Based on the dynamics of the plant and guidelines from [Åström and Wittenmark, 1984] the sample interval is 0.01 seconds (10ms).

Figure 3-5 shows the block diagram of PID control used to stabilize the inverted pendulum:

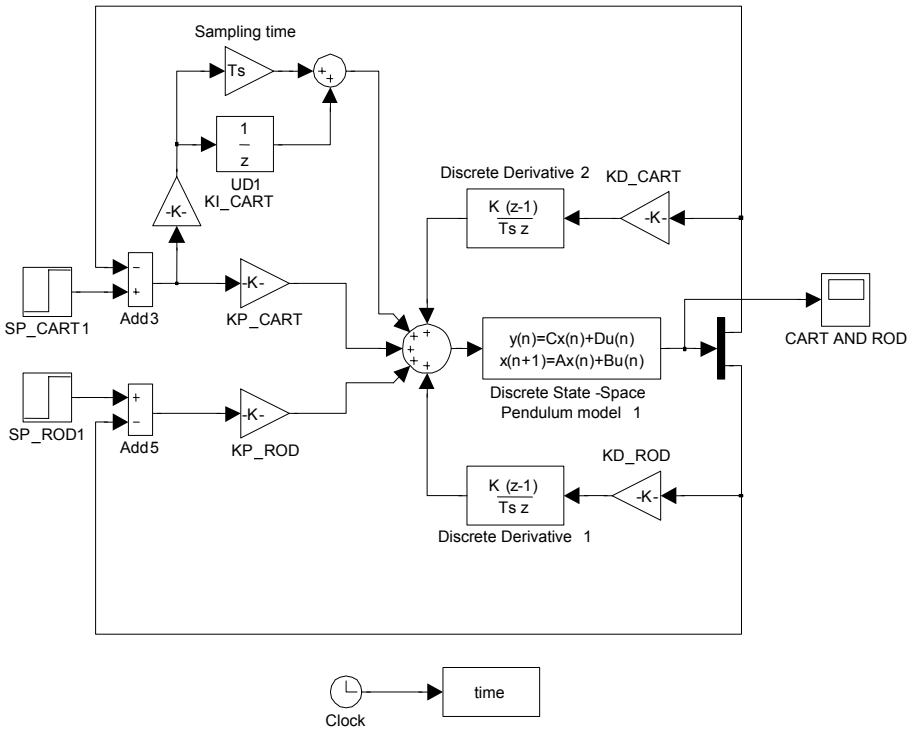

Figure 3-5 Inverted pendulum and PID controller block diagram. The cart is controlled using a PID action and the rod is controlled using a PD action. The set point for the cart is 0.3 metres and the set point to the rod is 0 radians (upright position).

For the experiment shown in the block diagram of Figure 3-5, the gains of the controller were K_{Pcart}=*-4.18 volts/metres,* K_{Icart}=*1.5 volts/metres seconds,* K_{Dcart}=*15.28 volts seconds/metres ,* K_{Prod}=*-31.38 volts/radians and* K_{Drod}=*7.574 volts seconds/metres*.

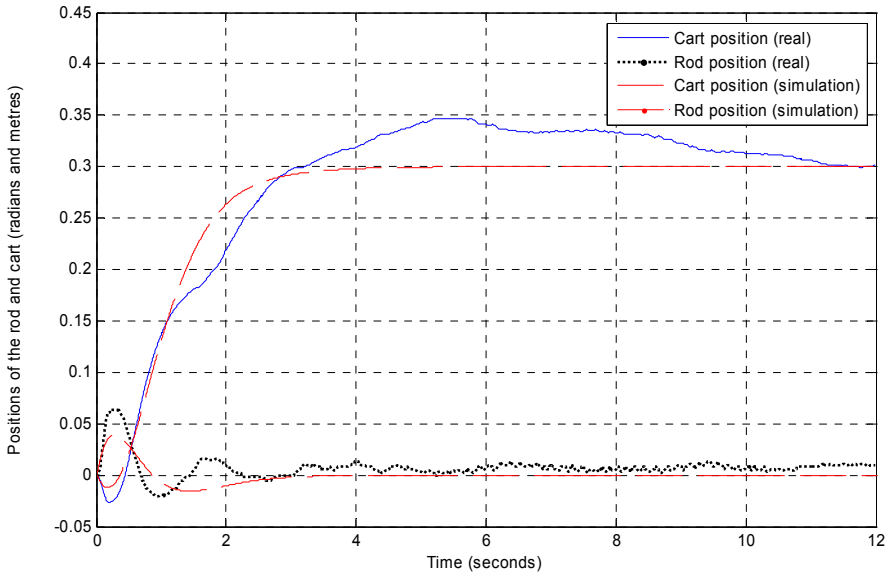

Figure 3-6: Step input response (position of the cart and rod) using PID algorithm, actual implementation.

The execution time of the PID control task is considered next: see Figure 3-7.

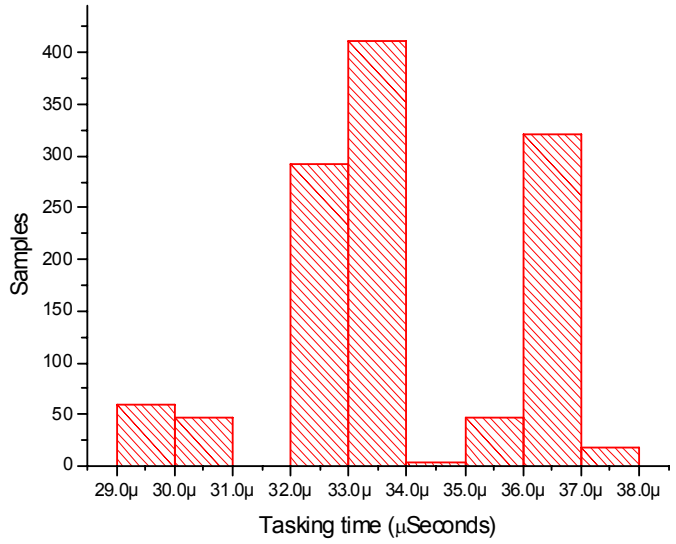

Figure 3-7: Histogram of the tasking time of the PID algorithm (Ts=10ms)

As an analysis of the results shows (Table 3-4), the execution time of the PID task is short (mean duration 30 µs in a sample time of 10,000 µs) and does not vary greatly (making it easy to schedule).

Table 3-4: Task timing statistics (PID algorithm)

Mean	*Standard Deviation*	*Standard error*	*Min*	*Max*	*Range*
33.8979µs	2.0002×10^{-6}	5.77411×10^{-9}	29.3667µs	37.7 µs	8.3 µs

Analysing more closely the interval between the actuation times (the actuation "jitter") for the PID algorithm the following results are obtained (Figure 3-8 and Table 3-5).

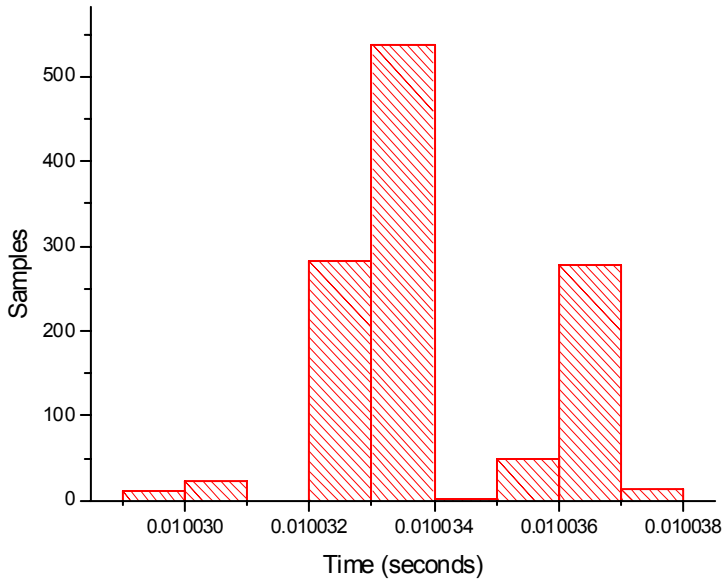

Figure 3-8: Histogram of the jitter for PID algorithm (Ts=10ms)

Table 3-5: Jitter statistics.

Mean	*Standard Deviation*	*Standard error*	*Min*	*Max*	*Range*
0.01003	1.66607μs	$4.80954e^{-8}$	0.01003	0.01004	8.36667μs

The range over which execution time varies is about 8 μs.

3.4.1 Implementation using different rod sensor resolutions

The numerical simulation and real-time signals of the cart's position and of the rod are compared in Figure 3-9 and Figure 3-10 . The lack of resolution in the rod sensor affects both the performance and stability of the system. Specifically, the plant behaviour becomes unpredictable when the 1000 ppr encoder is used to acquire the angular position of the rod.

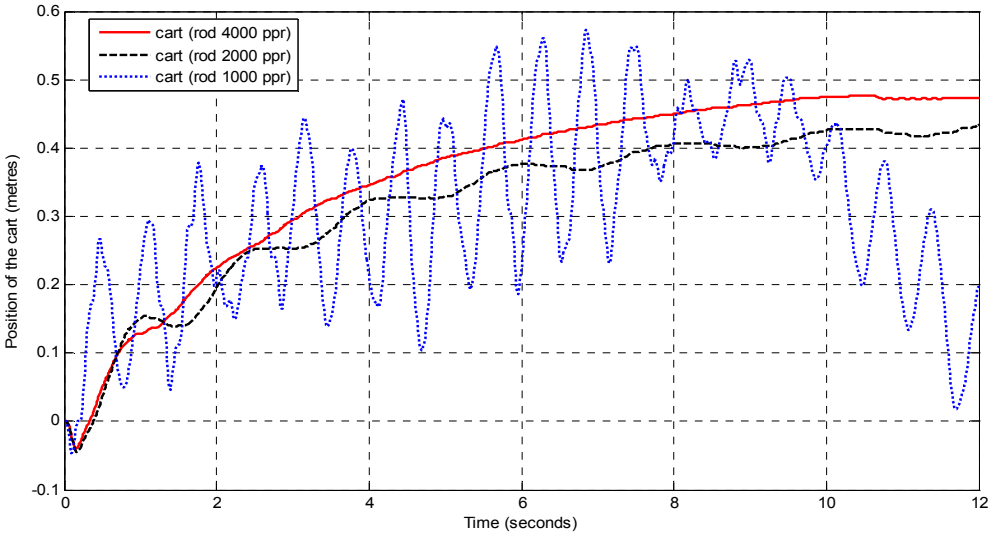

Figure 3-9 Cart plots using PID control with 4000, 2000 and 1000 ppr rod sensor resolution, actual implementation.

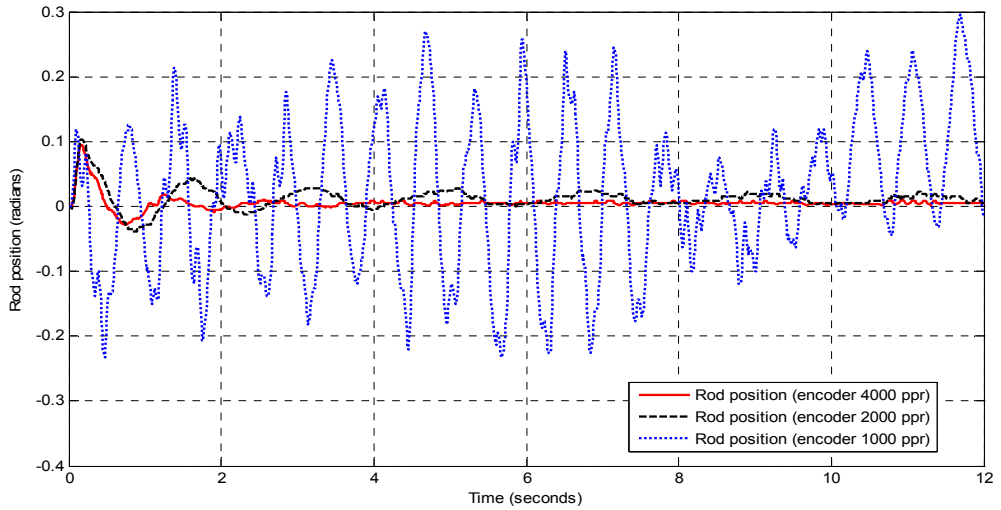

Figure 3-10 Rod plots using PID control with 4000, 2000 and 1000 ppr rod sensor resolution, actual implementation.

3.5 Choice of computing platform

The computing platform used in this research is a NXP LPC2129 microcontroller device based on a 32-bit ARM7 core that costs about $1 (US).

Instead of using an operating system (OS) as the software-architecture for the code to run this research employs a simple yet effective scheduler. This kind of software architecture is used in applications with severely resource-constrained control hardware. In this research, a co-operative scheduler was chosen for the features related to simplicity and predictability in the processing of the tasks. The programs were written in the C language for the microcontroller mentioned above.

As it was mentioned in the introduction, the scope of this research includes multiple control strategies applied to low-cost microcontrollers used in mechatronic systems. If the combination of different mechanical hardware, actuators, sensors and electronic interfaces are chosen randomly, the outcome would be catastrophic. On the contrary, this work combines strategies to select the proper hardware and software using optimisation criteria based on system performance and cost.

In the last decade, the market of embedded 32-bit processors started to narrow the performance gap between embedded and desktop systems [Schlett, 1998] Figure 3-11.

Nevertheless, more recently the market for 32-bit devices has been growing with a direct consequence that the cost for such processors has dropped substantially, see Table 3-6.

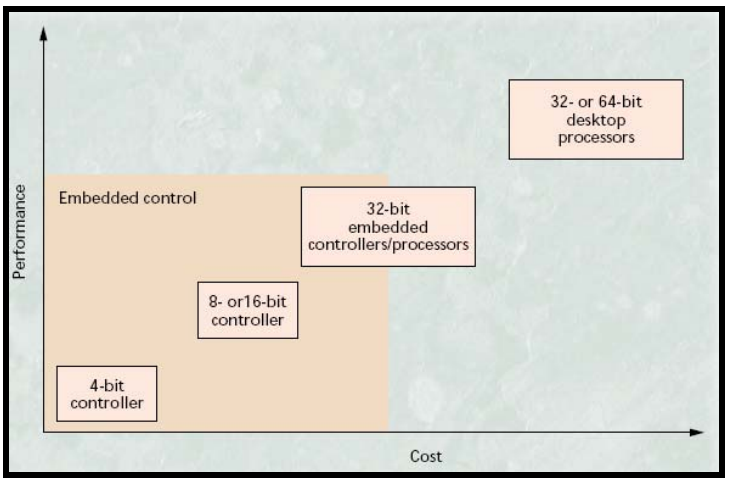

Figure 3-11 Performance and cost of different families of processors marking the boundaries of embedded control (source [Schlett, 1998])

Table 3-6 Average selling price for embedded processors (Source: [WSTS, 2005])

Device Type	*Average Selling Price*
Microprocessor, 32-bit	$92.89
Microprocessor, 16-bit	$6.87
Microprocessor, 8-bit	$3.72
Microcontroller, 32-bit	$7.57
Microcontroller, 16-bit	$4.23
Microcontroller, 8-bit	$1.44
Microcontroller, 4-bit	$0.78
Peripherals	$6.08
Digital signal processor (DSP)	$6.44
Average price for category	$6.12

Based on these facts, the applications in this research will focus at this level of small embedded computers of 32- bits that combine not only great computational power but also low cost.

For mechatronic applications in which floating-point operations are performed, 32-bit devices show important advantages over other architectures.

3.6 Conclusions

This chapter was meant to show the technical aspects of the testbed (dynamic model) and the benchmark controller (PID). It has shown all the theory elements

and practical knowledge to achieve a successful implementation. Moreover, the control hardware was selected in a trade-off manner between system functionality and overall cost.

Chapter 4
Supporting the implementation of FLC for ES

This chapter's focus is to explore the mechanism to implement FLC in resource-constrained embedded systems.

4.1 Introduction

The chapter[7] is divided in two parts. The first part shows an implementation of Fuzzy Logic Control (FLC) using simple heuristic rules. The second part includes an automatic learning method in order to tune the FLC algorithm. The inverted pendulum is used for the real-time experiments, in addition, simulation of two dynamic systems are presented.

4.2 FLC, implementation in embedded systems

Although principles of feedback control have been used for a long time, just in the last century many new approaches in this field caused an important technological impact in technology. Nevertheless there are a number of challenges that still require – for example – the development of theory for local output-input stability [Liberzon, 2003]. When dealing with a multi-variable, non-linear plant, etc. there are certain restrictions in the adoption of adaptability schemes [Polycarpou, 1996].

[7] Part of this chapter was presented in the paper: Bautista-Quintero, R. and Pont, M. J. "Is fuzzy logic a practical choice in resource-constrained embedded control systems implemented using general-purpose microcontrollers?" Proceedings of the 9th IEEE International Workshop on Advanced Motion Control, Volume 2, pp.692-697. IEEE catalogue number 06TH8850. ISBN 0-7803-9511-5

The emergence of "fuzzy logic" [Zadeh, 1965] and its introduction for non-trivial control applications [Yamakawa, 1992] were seen by many researchers as a way of avoiding the need for complex mathematical analysis and – instead – allowing the use of simple linguistic rules when developing complex control systems [Jinwoo et al., 1996]. However, in some applications, it has proved difficult to identify a set of linguistic rules [Alvarez et al., 1999]. To address this problem, self-learning inference methodologies have been developed. For example, Hiroyoshi Nomura in 1996 reported a novel technique based on an inference approach for an automatic learning fuzzy system [Nomura et al., 1996] used in industrial applications for Matsushita Electric Co. Another approach is based on adjustable fuzzy sets and rules under the principle of the on-line adaptive system [Wang, 1998]. A similar technique for auto-tuning FLC using neural networks was widely accepted [Jang, 1993].

All this effort to support the use of FLC has attracted the attention of industrial developers of embedded systems [Jackson, 1997]. In the field of research, there have been many previous contributions, in which "real" hardware has been implemented, for example: [Yamakawa, 1992], [Song et al., 2005], [Faa-Jeng and Po-Hung, 2006], [Magana and Holzapfel, 1998] and [Gaixin et al., 2002]. In some cases, contributions are based on simulation environments, including MATLAB and similar tools [Teng, 2000]. However, most studies fail to consider the cost of implementing and tuning the control algorithm (take into account optimisation of the CPU and memory usage). In this context, the computational resources (CPU and memory) are often greatly constrained when compared to equivalent desktop designs [Mrad et al., 2000].

The problem to implement FLC in resource-constrained systems is related to what has been called the "rule explosion" problem [Bellman, 1996]. This can be described as follows: if the system has n inputs and each input has m fuzzy sets (membership functions) it requires m^n rules. To deal with what may be a

significant rule set, it seems inevitable that it will require a fast processor with a large supply of memory: such requirements are incompatible with the resource constraints in many embedded designs.

In the next section, a simple FLC algorithm will be implemented using a heuristic rule design. A set of experiments will be conducted to show the memory and CPU performance on the embedded system employed.

4.3 The testbed

The ways in which an inverted pendulum may be used as an effective testbed by researchers who wish to explore different design options for reliable embedded systems have been previously described, see for instance, [Edwards et al., 2004] and [Bautista-Quintero et al., 2005]. One reason for selecting such a testbed is that this system is inherently unstable and provides a demanding control task, with a simple – clearly visible – indication of performance. In addition, as this is a well-studied control problem [Bishop and Dorf, 2004] and [Franklin et al., 1993], a great deal of useful information is available to support the development of such a testbed [Ogata, 2002].

The dynamic model of the testbed is shown in Chapter 3 and further information is given elsewhere [Bautista-Quintero et al., 2005].

4.4 Design and implementation of the FLC

As noted in the introduction, the rule explosion problem arises as follows: if the system has n inputs and each input has m fuzzy sets (membership functions), it requires m^n rules. For the work described in this Chapter, the number of inputs (n) is 4 and it was determined (using experience of an expert) that at least 5 (m = 5) fuzzy sets would be required: this means that the number of rules would be 625.

In order to implement the controller using the ARM processor, the rule set was "compacted". Specifically, greater priority was given to the rules that control the rod angle, with the control of the cart position treated as a lower priority. The process of selecting a compact set of rules was performed off line with a desktop computer running a C++ compiler and using experimental values obtained from the testbed. Similar rule reductions could be achieved using other techniques [Wang, 1998].

In Figure 4-1 illustrates how the resulting FLC controller was connected to the pendulum system.

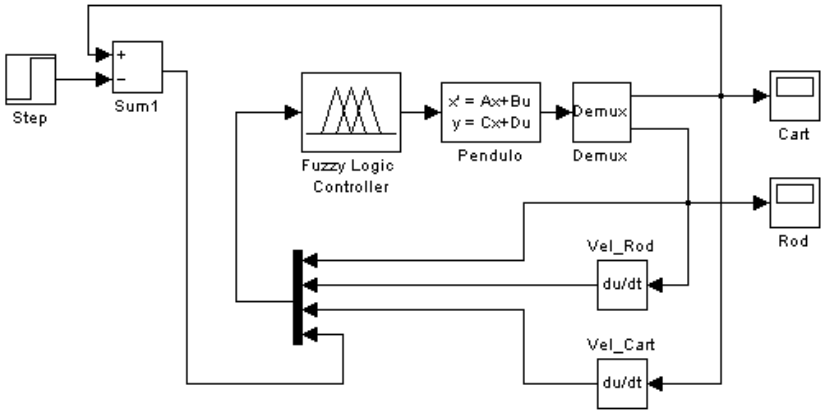

Figure 4-1: Block diagram of the inverted pendulum using a fuzzy logic algorithm

To make the operation of the fuzzy algorithm clear, Code 4-1 and Code 4-2 show "pseudo code" for the two subroutines used in the embedded system to "fuzzify" the real input variables (position and velocities of the cart and rod).

```
void fuzzy_inputs(float data_in,int num_var)
{
i=num_var;
crisp_in[i]=data_in;
for(j=0;j<num_input_mfs[i];j++)
 {
for(k=0;k<Coord_Points-1;k++)
  {xa=inmem_points[i][j][k];
   xb=inmem_points[i][j][k+1];
   if((xa<= crisp_in[i]) && (xb>=crisp_in[i]))
      {
      if(xa==xb)fuzzy_in[i][j]=1;
      else{ if(k==0)fuzzy_in[i][j]=(crisp_in[i]-xa)/(xb-xa);
          if(k==1)fuzzy_in[i][j]=1;
          if(k==2)fuzzy_in[i][j]=(crisp_in[i]-xb)/(xa-xb);
          }
       }
   }
  }
 }
```

Code 4-1: Crisp input into fuzzy world

In Code 4-1, the function *fuzzy_inputs* has two inputs (*data_in*, *num_var*). The use of this function is to transform the real value of the position and velocity of the cart and rod into "fuzzy variables". For reasons of simplicity, the fuzzy sets are based on trapezoidal and triangular shapes: the microcontroller requires only a 4-element array to represent such membership functions.

In Code 4-2, the function *rules_eval* calculates the minimum fuzzy value of each rule and the consequent is evaluated based on the maximum value of each strength rule. The output is calculated using a COG "defuzzification" method using a singleton-output membership function [Passino and Yurkovich, 1998].

```
float rules_eval(char set_num)
{
for(j=0;j<num_inputs;j++)
  {
   for(i=0;i<num_rules[set_num];i++)
    {
     a=rules[set_num][i][j];
     ante[j][i]=fuzzy_in[j][a];
     }
  }
for(i=0;i<num_rules[set_num];i++)
 {
 a=rules[set_num][i][num_inputs];
 conse[set_num][i]=outmem_points[set_num][a];
 }
  suma_num=0;
  suma_den=0;

 for(i=0;i<num_rules[set_num];i++)
  {
  min[i]=ante[1][i];
  for(j=0;j<num_inputs;j++)
     if(ante[j][i] < min[i])
min[i]=ante[j][i];
    sum_num=sum_num+conse[set_num][i]*min[i];
    sum_den=sum_den+min[i];
  }if(sum_den==0){return 0;}
return  sum_num/sum_den;}
```

Code 4-2 Function to evaluate fuzzy rules and calculate the output

```
float rules[num_sets][num_rules1][no_ante+no_conse]=
{
{
{ 2, 1, 2, 1, 2 },// ang zero & vel_ang zero then volta zero
{ 1, 1, 2, 1, 3 },// ang -    & vel_ang zero then volta +
{ 0, 1, 2, 1, 4 },// ang --   & vel_ang zero then volta ++
{ 3, 1, 2, 1, 1 },// ang +    & vel_ang zero then volta –
{ 4, 1, 2, 1, 0 },// ang ++   & vel_ang zero then volta –
{ 1, 0, 2, 1, 0 },// ang -    & vel_ang --   then volta –
{ 3, 2, 2, 1, 4 },// ang +    & vel_ang ++   then volta ++
{ 2, 0, 2, 1, 3 },// ang zero & vel_ang – then volta +
{ 2, 2, 2, 1, 1 },// ang zero & vel_ang + then volta –
{ 2, 1, 2, 1, 2 },// car zero & vel_car zero then volta zero
{ 2, 1, 1, 1, 3 },// car -    & vel_car zero then volta +
{ 2, 1, 0, 1, 3 },// car --   & vel_car zero then volta +
{ 2, 1, 3, 1, 1 },// car +    & vel_car zero then volta –
{ 2, 1, 4, 1, 1 },// car ++   & vel_car zero then volta –
{ 2, 1, 1, 0, 0 },// car -    & vel_car – then volta –
{ 2, 1, 3, 2, 3 },// car +    & vel_car + then volta +
{ 2, 1, 2, 0, 1 },// car zero & vel_car – then volta –
{ 2, 1, 2, 2, 3 } // car zero & vel_car + then volta +
}
};
```

Code 4-3: Set of rules to evaluate rules and calculate the output

The rules are expressed numerically in a three dimensional array (Code 4-3). At the system output, singleton membership functions are implemented in order to use the minimum amount of memory for the actuation function, see Code 4-5.

The step input response to the cart position is shown in Figure 4-2, where the simulation of the FLC (using a small set of rules) degrades its performance in the real system. Additionally, the overall performance of the real system compared with the result shown in Figure 3-6 (PID) performance is largely inconsistent with the original design presented (using PID control).

```
float inmem_points[num_inputs][num_max_MSF][Coord_Points] =
{ {
    { -30.000000, -30.000000, -10.000000, -5.000000 },
    { -10.000000, -5.000000, -5.000000, 0.000000 },
    { -5.000000, 0.000000, 0.000000, 5.000000 },
    { 0.000000, 5.000000, 5.000000, 10.000000 },
    { 5.000000, 10.000000, 30.000000, 30.000000 }
  },
{
    {-5.000000, -5.000000, -3.000000,0.000000 },
    { -3.000000, 0.000000, 0.000000, 3.000000 },
    { 0.000000, 3.000000, 5.0000000, 5.0000000 }
},
 {
    { -40.000000, -40.000000, -32.000000, -21.000000 },
    { -32.000000, -21.000000, -21.000000, 0.000000 },
    { -21.000000, 0.000000, 0.000000, 21.000000 },
    { 0.000000, 21.000000, 21.000000, 32.000000 },
    { 21.000000, 32.000000, 40.000000, 40.000000 }
 },
{
    {-40.000000, -40.000000, -20.000000,0.000000 },
    { -20.000000, 0.000000, 0.000000, 20.000000 },
    { 0.000000, 20.000000, 40.000000, 40.0000000 }
 }
};
```

Code 4-4: Rules and fuzzy sets used for the sub-actuated robot fuzzy control

```
float outmem_points[1][num_conse1] =
{
 {
  { -16.000000 },
  { -12.000000 },
  { 0.000000 },
  { 12.000000 },
  { 16.000000 }
 }
};
```

Code 4-5: Singleton membership function

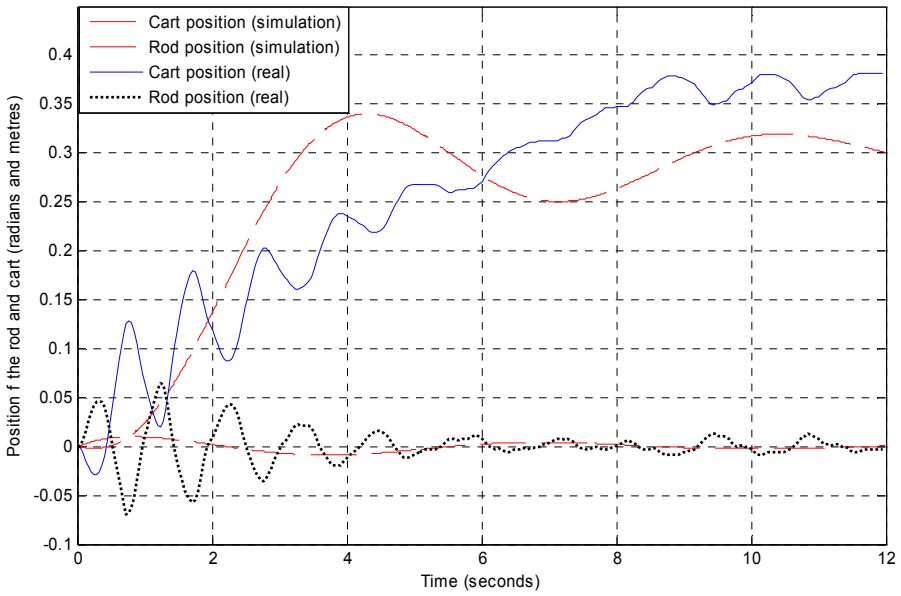

Figure 4-2 Step input response of fuzzy controlled implemented in ARM 7 microcontroller

The measurements of the task timing and actuation jitter for the FLC system are presented in Figure 4-3, and Figure 4-4.

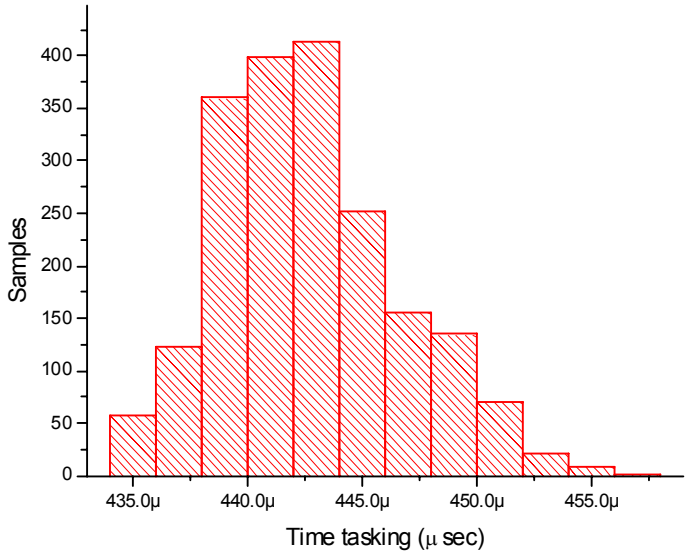

Figure 4-3: Histogram of the task timing of FLC algorithm.

Table 4-1: Task timing for the FLC algorithm

Mean	*Standard Deviation*	*Standard error*	*Min*	*Max*	*Range*
442.7µs	3.919 µs	$8.7x10^{-8}$	434.4 µs	456.2µs	21.8 µs

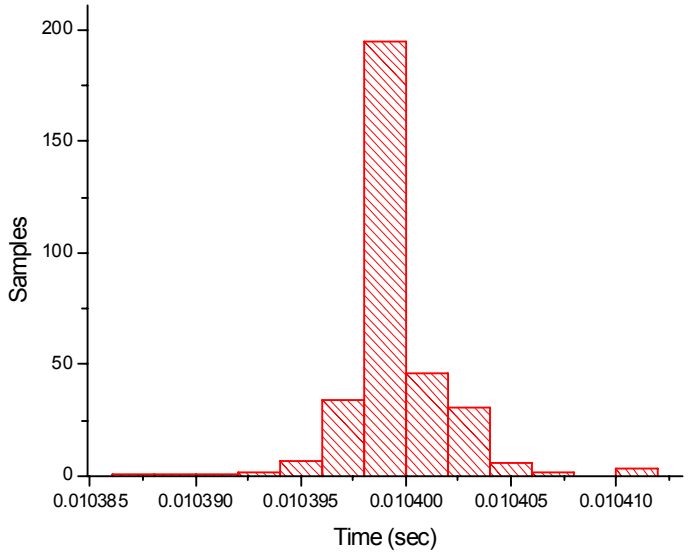

Figure 4-4: Jitter histogram of FLC algorithm

Table 4-2: Jitter statistics of the FLC algorithm

Mean	*Standard Deviation*	*Standard error*	*Min*	*Max*	*Range*
0.0104	2.439µs	$1.344x10^{-7}$	0.01039	0.01041	25.0µs

The task timing for this FLC controller that incorporates only 18 rules (625 rules constitute the complete set) averages 443 µs (compared with around 30 µs for the PID controller). There is also significantly more actuator jitter (in the test implementation): a range of 25 µs (compared with around 9 µs for the PID algorithm).

4.5 Comparing the algorithms

The differences between the two controller implementations are summarised in the Table 3-4.

Table 4-3: Comparing the two controller implementations

Parameter	*PID*	*Fuzzy*
Task timing (% of tick interval)	0.303%	4.42%
Actuator jitter (% of tick interval)	0.09%	0.25%
RAM memory requirement	1284 bytes	2251 bytes
Constants used	128 bytes	961 bytes
Flash memory program	4044 bytes	4740 bytes

Please note that, where specialist hardware (with support for FLC) is available, this is likely to have an impact on the results obtained.

4.6 Neuro-FLC approach for embedded systems

The difficulty of designing and developing reliable code using a heuristic FLC for embedded systems suggested that an implementation technique should be considered. A self-learning algorithm called Adaptive-Network-Based Fuzzy Inference System (ANFIS) approach was employed for this purpose.

ANFIS algorithm (described in [Jang, 1993]) automates FLC tuning using linear regression models to minimise the sum of the square errors (inputs-output map) between a functional controller and the FLC. However, the rule explosion problem limits implementation in low-cost processors.

There have been various attempts to deal with what may be a significant rule reduction in fuzzy-logic controllers. Some methods developed in 1969 address the problem using clustering methods [Ruspini, 1969]. Other methods have used neural network algorithms [Heckenthaler and Engell, 1994] which are based in probabilistic approaches that change according to the application. A technique for

rule reduction is described in [Rovatti et al., 1995] in which some conditions can be discarded by Boolean logic. Alternatively, a new technique presented more recently [Bezine et al., 2000] decouples a multivariable Fuzzy system in n subsystems in order to control them as a distributed control system. In this context, the rule explosion problem can be addressed by dividing the general system in sub-blocks.

Decoupling FLC implies a division in subsets of "n" controllers dedicated to maintain stable every subsystem. These controllers should be robust enough to control the local dynamic interaction in the whole plant. Since stability issues have not been presented, an empirical design is presented in order to reduce the number of MFs. In Figure 4-5 the decoupling approach is shown. The number of rules in the system is n^{MF} and by using the decoupling rule reduction algorithm it becomes: $\frac{n}{2}(2^{MF})$.

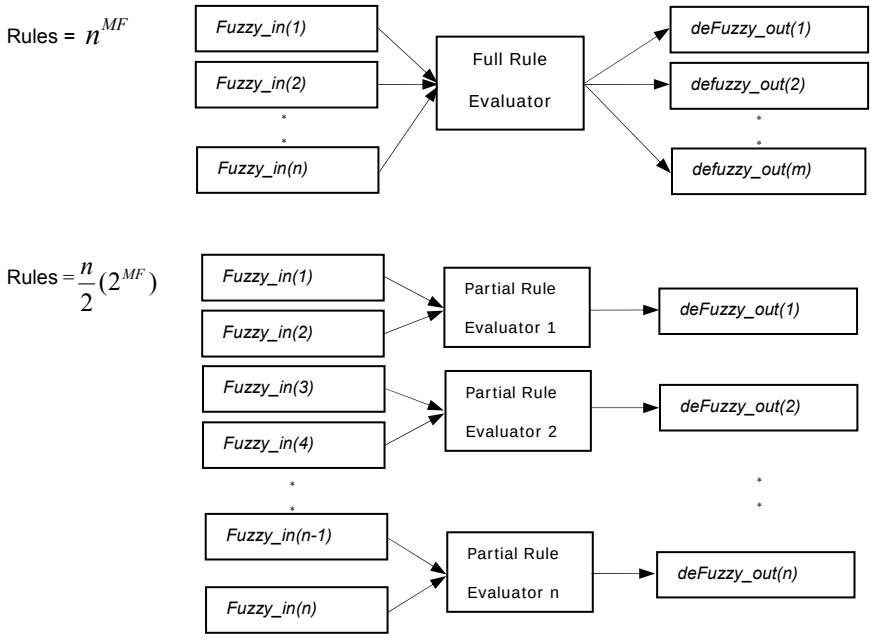

Figure 4-5 Comparing the traditional FLC structure of full rule evaluator system and the decoupled approach.

4.6.1 Decoupling example

A two-degrees-of-freedom (2-DOF) robot is presented to implement a decoupled ANFIS FLC algorithm. An independent controller is dedicated to provide stability to the position for each link. Interaction between links is considered a disturbance for each controller. So that, in order to assure that disturbances do not exceed the actuator limits, an estimation of this interaction between the links of the robot must be obtained. A dynamic model is used to perform such estimations.

The dynamic model is taken from [Reyes and Kelly, 2001] using Euler-Lagrange equation see Equation 4-1. The 2-DOF robot is shown in Figure 4-6

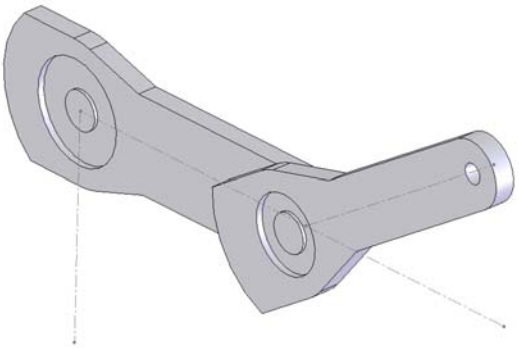

Figure 4-6 Two degree of freedom Robot prototype developed by R. Kelly, see [Kelly and Santibanez, 2003].

$$\begin{bmatrix} M_{11}(q) & M_{12}(q) \\ M_{21}(q) & M_{22}(q) \end{bmatrix} \ddot{q} + \begin{bmatrix} C_{11}(q,\dot{q}) & C_{12}(q,\dot{q}) \\ C_{21}(q,\dot{q}) & C_{22}(q,\dot{q}) \end{bmatrix} \dot{q} + \begin{bmatrix} G_1(q) \\ G_2(q) \end{bmatrix} = \tau \qquad \text{Equation 4-1}$$

where:

$$M_{11}(q) = m_1 l_{c1}^2 + m_2\left[l_1^2 + l_{c2}^2 + 2 l_1 l_{c2}\cos(q_2)\right] + I_1 + I_2$$
$$M_{12}(q) = m_2\left[l_{c2}^2 + l_1 l_{c2}\cos(q_2)\right] + I_2$$
$$M_{21}(q) = m_2\left[l_{c2}^2 + l_1 l_{c2}\cos(q_2)\right] + I_2$$
$$M_{22}(q) = m_2 l_{c2}^2 + I_2$$
$$C_{11}(q,\dot{q}) = -m_2 l_1 l_{c2}\sin(q_2)\dot{q}_2$$
$$C_{12}(q,\dot{q}) = -m_2 l_1 l_{c2}\sin(q_2)\left[\dot{q}_1 + \dot{q}_2\right]$$
$$C_{21}(q,\dot{q}) = m_2 l_1 l_{c2}\sin(q_2)\dot{q}_1$$
$$C_{22}(q,\dot{q}) = 0$$
$$G_1(q) = \left[m_1 l_{c1} + m_2 l_1\right] g\sin(q_1) + m_2 l_{c2} g\sin(q_1 + q_2)$$
$$G_2(q) = m_2 l_{c2} g\sin(q_1 + q_2)$$

Where the parameters of the robot are shown in Table 4-4.

Table 4-4 Robot parameters (taken from [Kelly and Santibanez, 2003])

Description	Notation	Value	Units
Length of link 1	l_1	0.45	M
Length of link 2	l_2	0.45	m
Distance to centre of mass (link 1)	l_{c1}	0.091	m
Distance to centre of mass (link 2)	l_{c2}	0.048	m
Mass of link 1	m_1	23.9	kg
Mass of link 2	m_2	3.88	kg
Inertia of link 1	i_1	1.266	kg m^2
Inertia of link 2	i_2	0.093	kg m^2
Gravity acceleration	g	9.81	m/s^2

Due to the high requirements in memory to implement a FLC, the decoupling technique offers important reduction in terms of processing time and memory. Implementation of such technique needs two decoupled fuzzy controllers (one for each link). The interaction torque (force) between the robot link1 due to link 2 is shown in Equation 4-2

$$Inte_{\tau_1} = m_2\left[l_{c2}^2 + 3 l_1 l_{c2}\cos(q_2)\right] + I_2 - 2 m_2 l_1 l_{c2}\sin(q_2)\dot{q}_2 + m_2 l_{c2} g\sin(q_1 + q_2) \qquad \text{Equation 4-2}$$

Now the torque (force) interaction of link 2 due to link 1 is shown in Equation 4-3

$$Inte_{\tau_2} = m_2 l_1 l_{c2} \sin(q_2)\dot{q}_1 + m_2 l_{c2} g \sin(q_1 + q_2) \qquad \text{Equation 4-3}$$

Having equations Equation 4-2 and Equation 4-3 the interaction among the links can be plotted in order to estimate disturbances for each link. In Figure 4-7 the level of interaction among the direct torque effect and the interaction torque is shown.

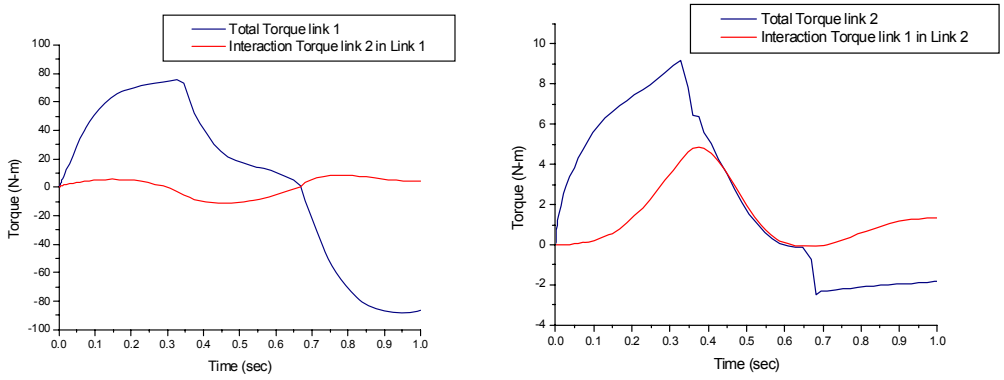

Figure 4-7 Relation of the total load torques against the interaction between links.

For this simulation, the dynamic model was tested in close loop with a PID controller using the following gains: *Kp_1=350 N-m/rad, Ki_1=50 N-m/seconds radians, Kd_1=35 N-m-seconds/rad, Kp_2=350 N-m/rad, Ki_2=50 N-m/seconds and Kd_2=35 N-m seconds/rad*.

Using an ANFIS algorithm a FLC is trained in order to adjust the MF of the controller, see Figure 4-8. The trajectory used to train the system was a linear segment with parabolic bends (LSPB) using an acceleration path of 9.03 rad/s^2 with a maximum speed of 6.28 rad/s.

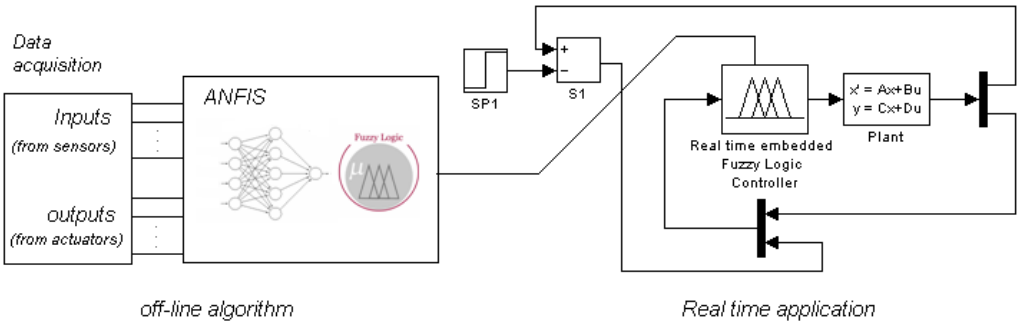

Figure 4-8 FLC trained using an ANFIS algorithm based on real-time acquisition. The acquisition is taken from the system in close loop using an auxiliary algorithm to teach the FLC.

For comparison purposes, the control system without the decoupling technique has a four input system (positions and velocities of links 1 and 2) and two outputs (actuators of links 1 and 2). Using seven Gaussian MFs, the FLC system has 2401 rules for each output.

The system's response for a step input is shown in Figure 4-9, the two links are meant to reach 1 radian.

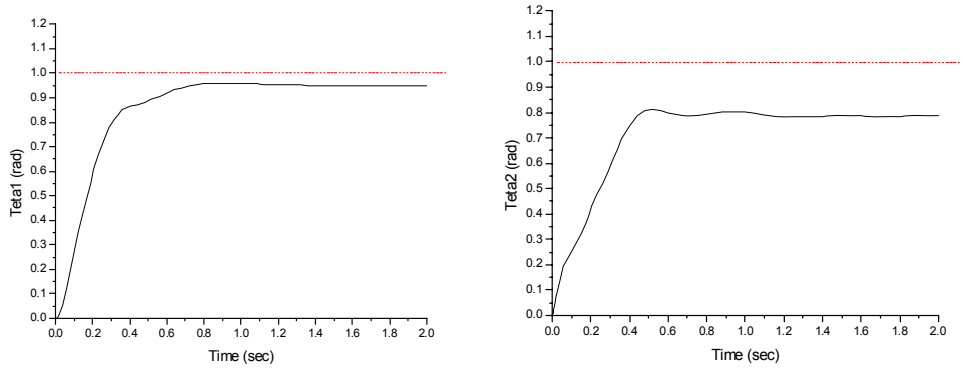

Figure 4-9 System step-response (q_1 –teta$_1$– and q_2 –teta$_2$–) using a FLC trained by an ANFIS algorithm.

The structure of the FLC controller and plant interconnection is shown in Figure 4-10. Positions and velocities of each link are fed back to the control system. The response is poor in terms of performance and the number of rules makes implementation in small resources impractical. By contrast, when the system is

divided to two FLC trained with the same methodology an important reduction in memory and execution time can be obtained. See Figure 4-11.

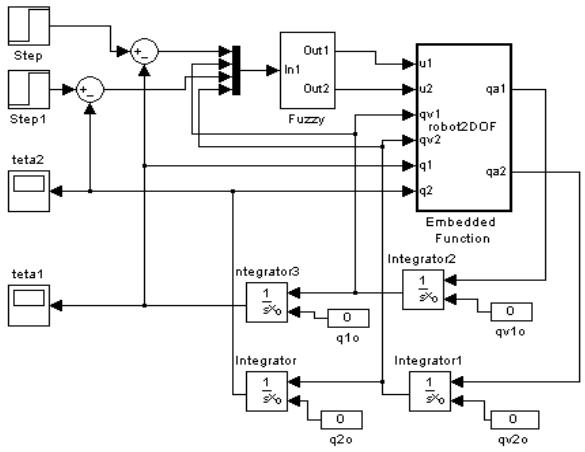

Figure 4-10 Structure of controller and plant using a full set of rules.

The decoupling approach reduced the number of rules and improved the system's performance.

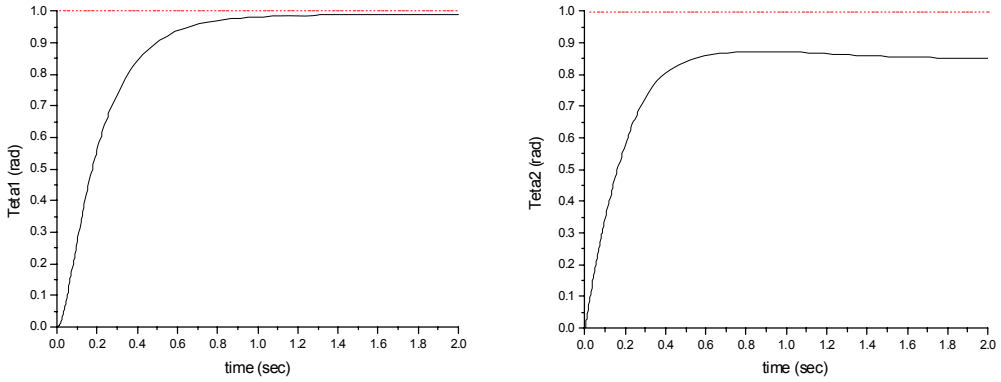

Figure 4-11 System step-response (q_1 –$teta_1$– and q_2 –$teta_2$–) using a decoupled FLC trained by an ANFIS algorithm.

Figure 4-12 shows the structure of the decoupled controllers, which has two inputs (position and velocity of each link) and one output each actuator (u_1 and u_2). Considering four MF in each controller the system has 32 rules to control the whole plant.

4.6.2 Implementation guidelines

A method to use these tools (FLC, ANFIS, decoupling-control) can be summarised in the flow diagram shown in Figure 4-13. It is a practical way to identify the number of rules required according to the resources available.

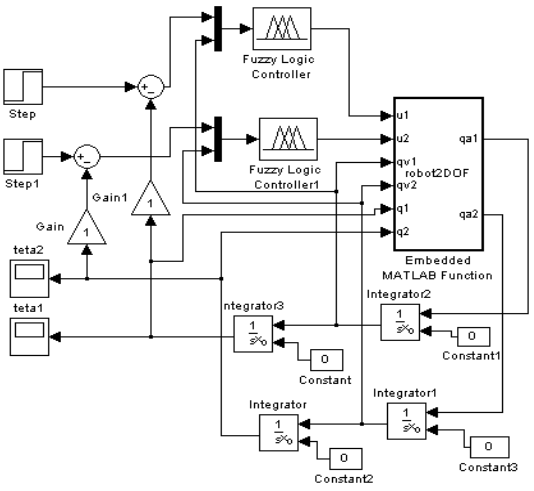

Figure 4-12 Structure plant and decoupled controlled (2 inputs, 1 output each, 4 triangular MF, 32 rules), where the non-linear model of the robot is in the box robot2DOF.

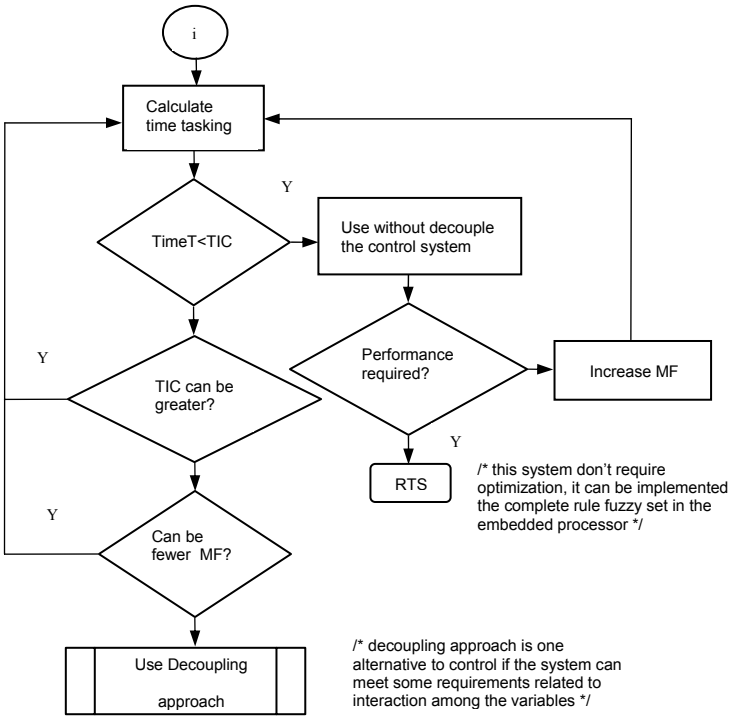

Figure 4-13 Flow diagram of the rule optimisation process.

4.7 Case study 1 (Servomotor control)

Servomotor position control is one of the most common tasks demanded in industry [Dote, 1990], [Godfrey, 2005], [Chang et al., 2001]. New approaches to make easier tuning and improve robustness for hostile environments have been developed. This case study presents a method to implement a FLC using a low-cost and off-the-shelf microcontroller. A numerical model is presented for validation purposes.

The model of the servomotor and load is presented in Equation 4-4. The variables are the velocity and position of the shaft, the external torque disturbance is (T_{ex}). the motor constant K_m (Volts/rad/s), the stator resistance R (Ohms), the motor inductance L (Henry) and the inertia moment J_m (Kg m^2). The states are the

velocity of the motor and armature current the inputs are armature voltage and external torque disturbance.

$$\dot{x} = \begin{bmatrix} 0 & \frac{K_m}{J_m} \\ -\frac{K_m}{L} & -\frac{R}{L} \end{bmatrix} x + \begin{bmatrix} 0 & -\frac{1}{Jm} \\ \frac{1}{L} & 0 \end{bmatrix} u \qquad \text{Equation 4-4}$$

A block diagram of the system is shown in Figure 4-14.

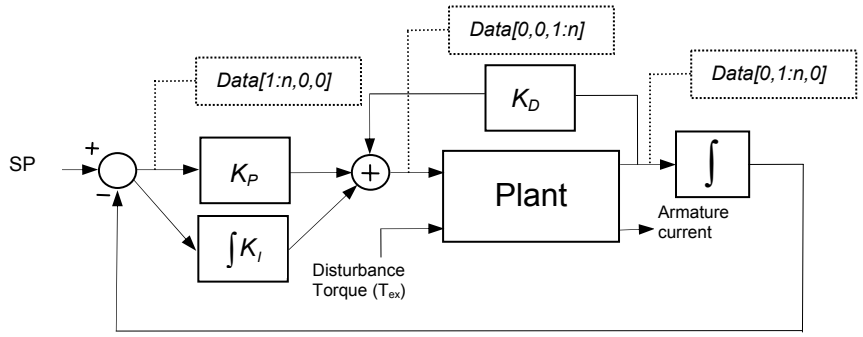

Figure 4-14 Model of DC motor with mechanical load with position and velocity feedback. Data [1:n,0,0], Data[0,1:n,0] are the input vectors used for training, and Data[0,1:n,0] output vector for the training.

A PID controller is employed to train the FLC. The vector *Data* (where inputs are data[1:n,0,0] and data[0,1:n, 0] and outputs are data[0,0,1:n]).

Following the flow diagram shown in Figure 4-13 the suitable sample period is given according to a specific number of MF are chosen in order to implement the system.

Once the system is trained the implementation was tested in the real plant, see block diagram Figure 4-15.

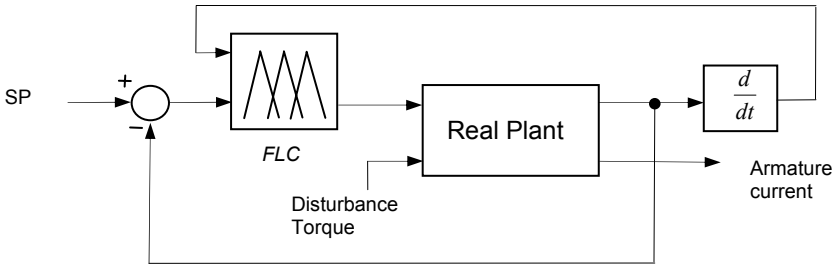

Figure 4-15 Actual implementation diagram of the FLC for two-inputs one-output system.

The FLC block contains the information to control the plant along the range of movement (previously trained). A comparison between PID and Neuro-FLC (N-FLC) showed the efficiency for this simple dynamic system. A step input response is shown in Figure 4-16. The performance of the system using FLC is similar to the PID design.

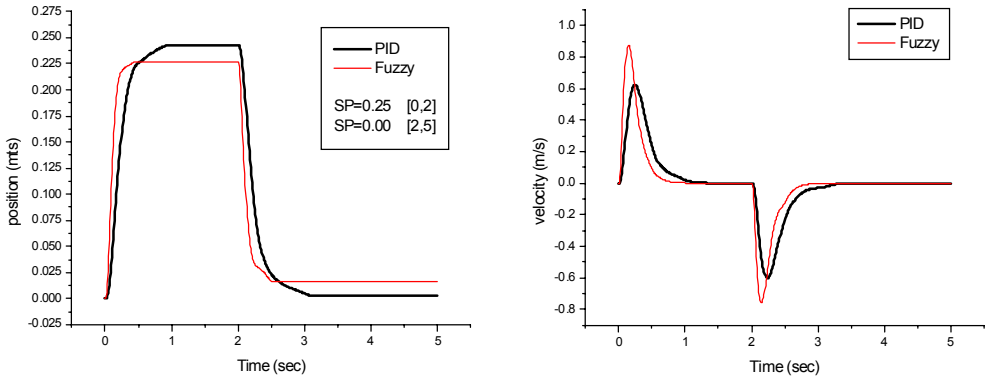

Figure 4-16 Actual implementation of the DC motor, it shows the position and velocity responses using both PID and FLC controllers. The set-point of 0.25 metres in the interval 0 to 2 seconds and 0 metres from 2 to 5 seconds.

Table 4-5 comparing the resources of memory and task timing in the testbed.

Element and resource	*PID*	*FLC*
Average execution-time	20.5 x10^{-6} s	82.8 x10^{-6} s
RAM (variables in program)	1276 Bytes	1350 Bytes
FLASH (Program memory)	3156 Bytes	3556 Bytes
Constants used in the program	112	192

For this system, 16 rules were required and memory requirements are shown in Table 4-5.

4.8 Case study 2 (Inverted pendulum)

For this case study, two controllers were used; one dedicated to control of the cart dynamics, and the other the rod dynamics. The FLC is tuned using a PID system described in the previous chapter.

As in the previous case study, position and velocity are fed back to the controller and the two FLCs provide the control signal to the plant (see Figure 4-17).

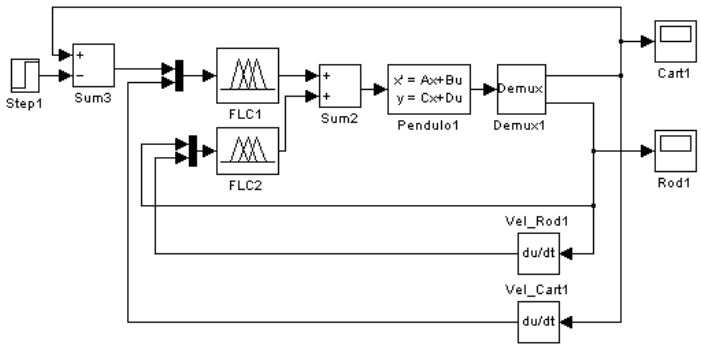

Figure 4-17 Distributed control using two FLCs for a system with four inputs and one input

For this case study, four MFs were used for each FLC, having 32 rules in total. In order to use the minimum resources while keeping the plant stable, the guidelines presented in the flow diagram shown in Figure 4-13 were followed.

Although the FLC tuned MF based on the PID controller, the system still showed important limitations in its ability to maintain a robust performance in presence of actual uncertainties. See the comparison between the two approaches in Figure 4-18.

The resources required to implement the N-FLC approach simplify enormously the empirical rule selection and tuning process. Additionally, using the decoupling technique, an important rule reduction is achieved. However, for this application (inverted pendulum), uncertainties caused for the inherent noise in the acquisition are difficult to parameterized during the self-learning algorithm. That limits the performance compared with the original controller (PID).

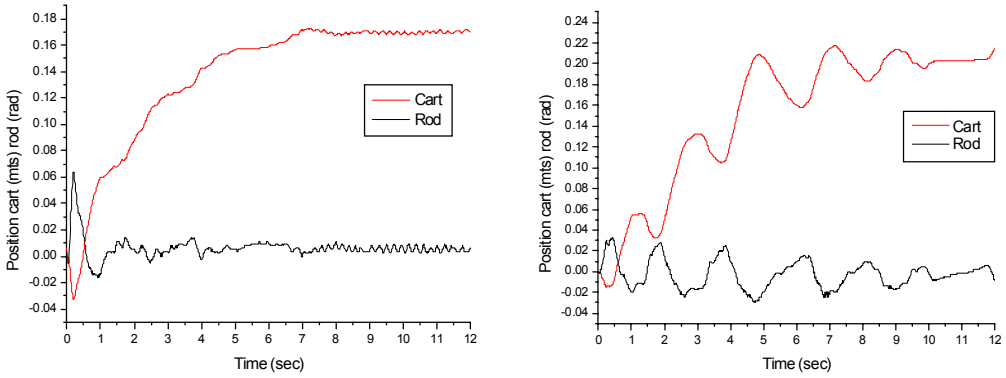

Figure 4-18 Actual control implementation of the pendulum, using the auto-learning approach. Step response to the system using PID control (left) and FLC (right).

Table 4-6 Comparing the resources of memory and task timing in the testbed

Element and resource	*PID*	*FLC*
Average execution time	32 x10^{-6} s	176 x10^{-6} s
RAM (program data)	1284 Bytes	1480 Bytes
FLASH (program code)	3952 Bytes	4480 Bytes
Constants used in the program	128	300

4.9 Conclusions

In this chapter, the performance and resource requirements of a "conventional" (PID) controller and a fuzzy logic controller were compared, using an inverted pendulum testbed. In each case, the controller was implemented on a low-cost microcontroller with limited CPU and memory resources. The results obtained suggest that, as expected, the resource requirements of the FLC design outweigh those of the "conventional" controller in this study. It is suggested that – where

embedded systems have severe resource constraints and “off the shelf” microcontrollers are used – fuzzy control (at least in the form implemented here) is unlikely to be a practical option for non-trivial control systems.

N-FLC was then considered as a means of alleviating the problems of tuning the controller and the lack of performance.

Numerical simulation of the self-learning algorithm shows important advantages in terms of simplicity of design and performance. Nevertheless, the case studies suggested that FLC is limited in terms of memory, execution time and performance when compared with PID.

Chapter 5
Assessing resource requirements for the optimal LQR

This chapter explores the linear quadratic regulator (LQR) technique and its implementation requirements using a resource-constrained embedded system.

5.1 Introduction

As discussed in the literature review (Chapter 2), LQR control emerged as an alternative algorithm in embedded systems in order to improve robustness and performance [Gully and Coleman, 1981]. The theory behind LQR deals with minimisation of a cost function [Bishop and Dorf, 2004]. In this chapter, an appropriate state-feedback controller is proposed to optimise the energy used in the plant controller (inverted pendulum). The goal is to show an analytical approach combined with the techniques to implement a robust control algorithm in resource-constrained embedded system. The particular focus of the chapter is on the LQR control algorithm and its comparison with the classic PID (showed in Chapter 3)[8].

This chapter employs an extended and modified version of the inverted pendulum testbed [Edwards et al., 2004] in order to explore the impact of different control algorithms on the system performance and resource requirements.

[8] Some of the elements presented in Chapter 5 have been published in R. Bautista-Quintero *et. al.* "Comparing the performance and resource requirements of "PID" and LQR algorithms when used in a practical embedded control system: A pilot study", Proceedings of the UK Embedded Forum, October 2005.

5.2 The testbed

As mentioned in Chapter 3, the inverted pendulum testbed is a system inherently unstable and provides a demanding control task, with a simple – clearly visible – indication of performance. The present work employs an extended and modified version with an incremental sensor (with simple, double and quad sensor resolution routines) and interchangeable mechanical hardware (mass and length of rod) in order to explore the impact of different control algorithms on the system performance and resource requirements.

During the course of the chapter, there are two main comparisons: the performance and robustness of these two different control algorithms when the mass and length of the rod is changed. The frequency response of the two algorithms is determined and it is compared with results obtained through the numerical simulations. Resources required for both algorithms are also presented for comparison purposes.

5.3 Design of an LQR controller

When the dynamic model of the system is available, it is possible to design a state-feedback controller. Modern control based on state-space methods (such as the LQR) is explored. This control algorithm is commonly used for multivariable linear systems [Chen, 1999]. For this particular problem, LQR is appropriate since the aim is not only to maintain the pendulum in an upright position but also to control the position of the cart. Additionally, the system control aims to minimise the energy consumed for the actuator. This method uses a cost function based on the solution of the differential *Riccati* equation (see [Franklin et al., 1993] for more details).

In Equation 5-1, Q and r that are positive-definite matrices. The second term of this equation represents the expenditure of the energy of the control signal. The

matrices Q and r determine the relative importance of the error and the expenditure of this energy, however the formulation assumes that vector $u(t)$ is unconstrained. Consequently, in a practical implementation, it is important to make a trade-off between the energy used and the performance of the system.

Vector x consists of four states (position of the cart x_1, velocity of the cart x_3, position of the rod x_2, velocity of the rod x_4) multiplied by the gain vector (k) to provide the input voltage. Therefore the control law $u=-kx$ is employed in order to minimise the quadratic cost function to the form:

$$J = \frac{1}{2}\sum_{k=0}^{\infty}\{x(k)^T Qx(k) + ru(k)^2\} \qquad \text{Equation 5-1}$$

For this application, the goal is meant to minimize not only the energy used in the system but also keep the system stable with effective robustness. The system is stable and the current that flows in the motor is minimal according to the cost function.

Q and r are positive-definite and are used to define the weights of the states (variables, positions and velocities of the cart and rod).

The relationship between the elements in the diagonal of the Q matrix represents the relative importance of the four terms (position and velocities of the cart and rod) to the control law. For the first approach, the same value was chosen in each "slot". As far as r is concerned, altering the value can change the speed of response of the controlled system (larger values of r make the response slower).

The values shown in Equation 5-2 were chosen according the numerical simulator (MATLAB). The aim was to reach the goal in the shortest rise-time without exceeding the maximum actuator limit. For this particular system the actuator limit is V_{max} ($u(t) < V_{max}$ =19.1 volts).

$$Q = diag[120,350,10,12]$$

Equation 5-2

$$r=0.1$$

Based on the discrete model and using an LQR controller, the gains for the linear version around equilibrium point are, k_1=-4.18 volts/radians, k_2=-33.27 volts/radians, k_3=-18.09 volts-seconds/radians and k_4=-7.89 volts-seconds/radians.

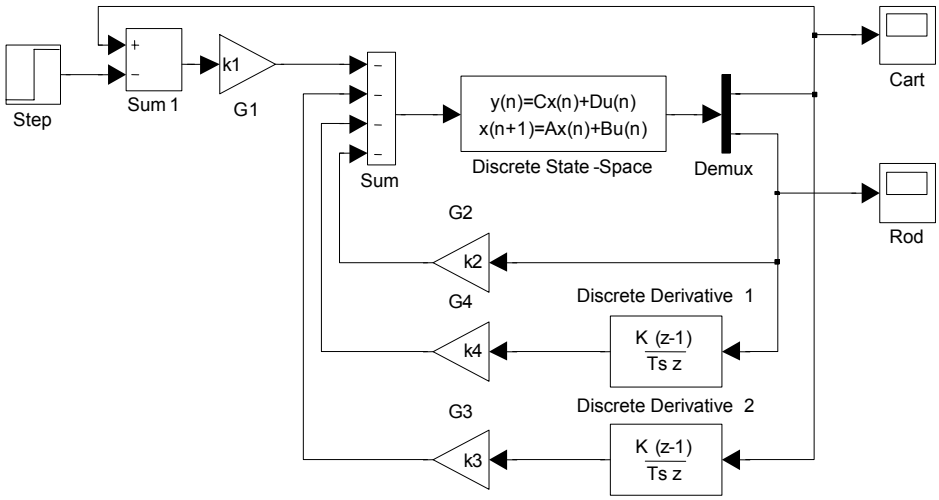

Figure 5-1: Block diagram of LQR controller and testbed

5.4 Implementation of an LQR controller

In this implementation of the LQR algorithm, the microcontroller obtains readings directly from two states from the system (the cart position and the rod angle). In addition, it approximates two states (velocity of the cart, velocity of the rod).

To determine the velocity of the rod, its position is measured periodically (at known time intervals, in this case, every 4 ms) and used as a simple "pseudo derivative" (first order derivative) to approximate the velocity. Then a low-pass filter is implemented to reduce the impact of the high frequency harmonics

generated in this approximation. In this case, the algorithm employed is the moving-average filter presented in Section 3.3.1 An identical procedure was followed when calculating the cart position.

In order to determine the control feedback gains, the discrete function for linear quadratic regulator (*dlqr*[9]) is used (based on the discrete model of the pendulum).

Figure 5-2 shows the step response of the resulting implementation.

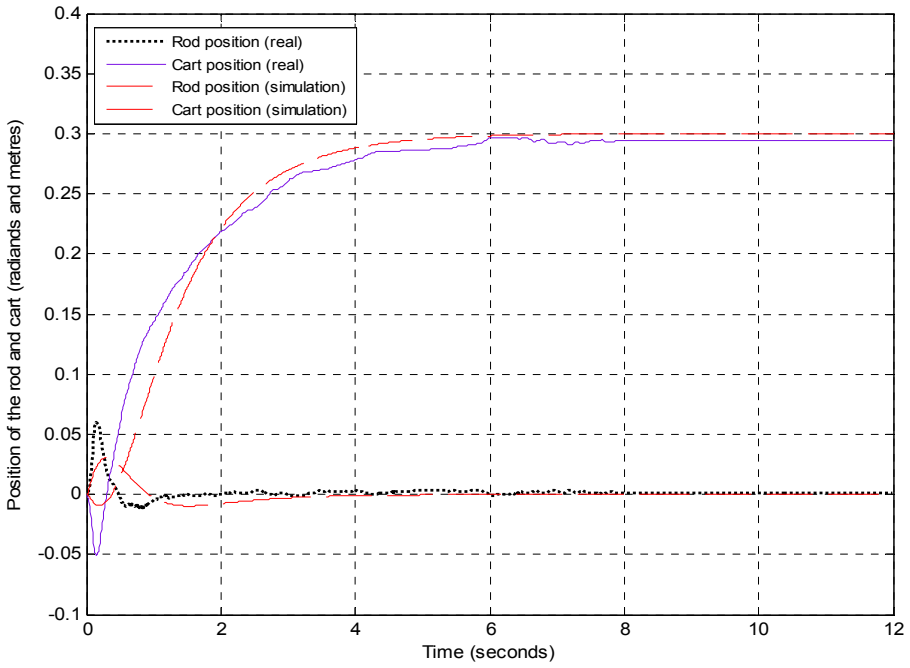

Figure 5-2 Actual implementation of the system using LQR controller, a step input for the cart position is set (SP_{cart} = 0.3 metres).

The PID control implementation is shown in Section 3.4 and it is compared in the next section.

[9] MATLAB function

5.5 Comparing the simulations and system implementations

In this simple comparison, some features are measured from the step response given from the numerical simulation and the real-time implementation. The data is presented in both Table 5-1 and Table 5-2.

Please note that – based on this comparison - LQR seems to be both more accurate and faster than PID. However, in situations with high frictional forces (higher than in the present testbed), the ability of PID control to meet the set point may be particularly advantageous (see Figure 3-6).

Table 5-1: Features of the system using LQR with a SP=0.3 metre.

	Simulation		***Testbed***	
Feature	For the cart:	For the rod:	For the cart:	For the rod:
Undershot	12.32 mm	0 rad	-50.8 mm	0.042 rad
Overshot	0 mm	0.025 mm	0 mm	0.057 rad
Steady state error	0 mm	0 mm	10.49 mm	0.01157 rad
Rising time	4 s	-	99% at 4.5 s	-

Table 5-2: Features of the system using PID with a SP=0.3 metre.

	Simulation		***Testbed***	
Feature	For the cart:	For the rod:	For the cart:	For the rod:
Undershot	11.9 mm	0 mm	27.3 mm	0.046 rad
Overshot	0 mm	0 mm	50.1 mm	0.037 rad
Steady state error	0 mm	0 mm	0 mm t=20s	0.0092 rad
Rising time	2 s	-	3.2 s	-

5.6 Comparing the basic performance of the two controllers

The performance of the two controllers in terms of dynamic set-point is compared in this section. One effective way of comparing the performance of different control algorithms is to consider the response of the controlled system to a

dynamic set point [Reyes and Kelly, 2001]. In the comparisons described in this section, the set point (SP) used is specified as follows:

$$SP = A\sin(wt) \qquad \text{Equation 5-3}$$

The initial frequency used was 1 Hz and this was reduced to 0.5 Hz, 0.25 Hz, 0.0125 Hz, 0.0625 Hz, and finally 0.03125 Hz. Identical tests were performed for the LQR and PID controllers and the results are presented in this section.

Overall, if PID is used in SISO systems the design may require simple rules for tuning. However, the MISO approach presented in this chapter (which requires several objectives to perform simultaneously) LQR control may be a more appropriate choice. In this context, the frequency response of the two algorithms shows important results to be discussed.

As can be seen from Figure 5-3 and Figure 5-4, the system response controlled by PID has more amplitude attenuation due to the frequency than the plant using LQR.

Additionally, the system under PID control tries to eliminate the effect of a sub-harmonic frequency presented in the experiment shown in Figure 5-4.

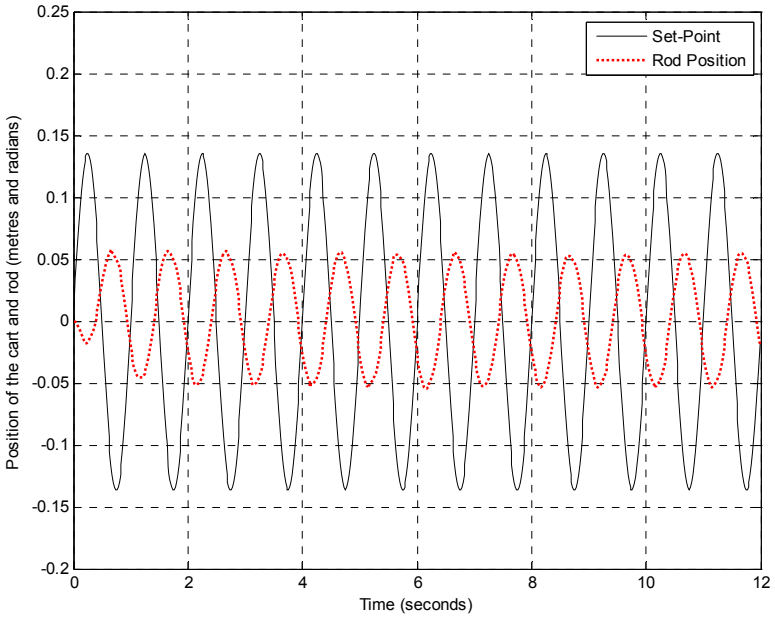

Figure 5-3: LQR Tracking control using a dynamical SP=0.135 sin (wt), Frequency=1Hz.

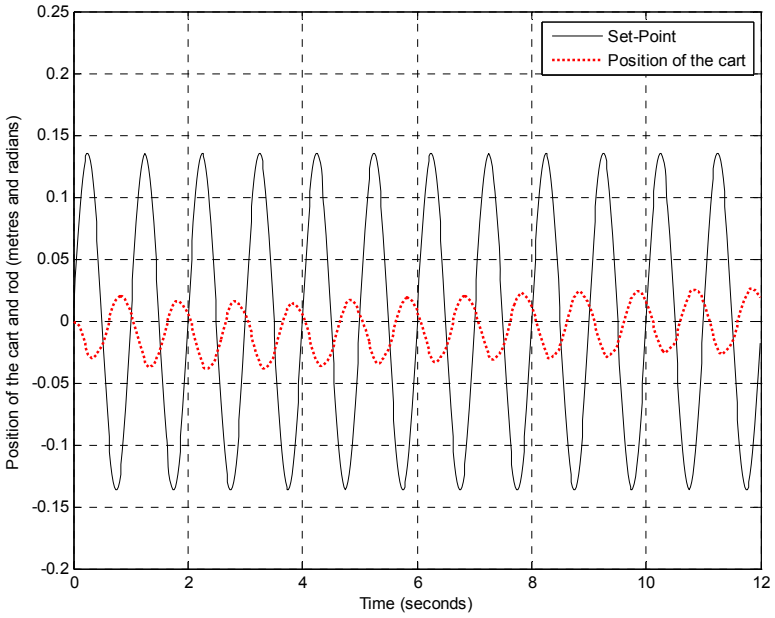

Figure 5-4: PID Tracking control using a dynamical SP=0.135 sin (wt), Frequency=1Hz.

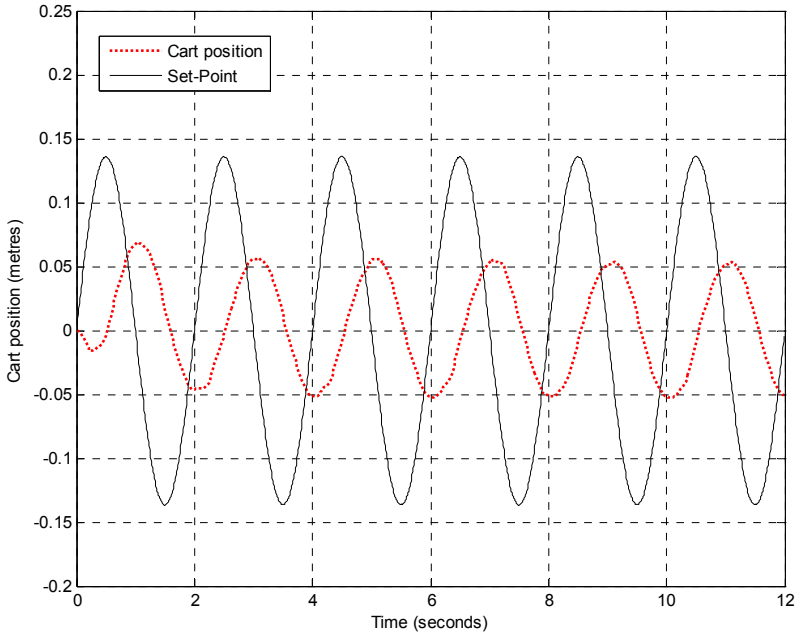

Figure 5-5: LQR Tracking control using a dynamical SP=0.135 sin (wt), Frequency=0.5Hz.

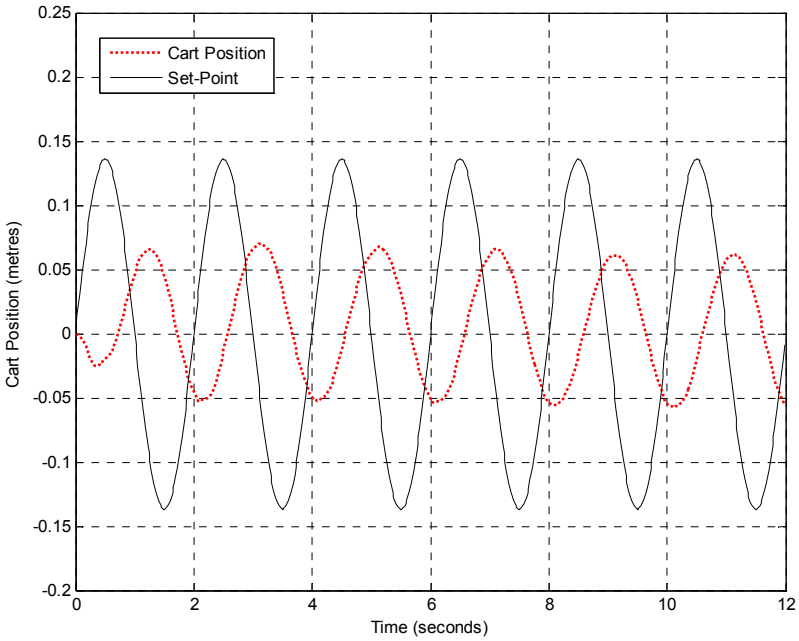

Figure 5-6: PID Tracking control using a dynamical SP=2000 sin (wt), Frequency=0.5Hz.

With a 0.5 Hz signal (Figure 5-5 and Figure 5-6), PID and LQR systems have the same attenuation and phase delay.

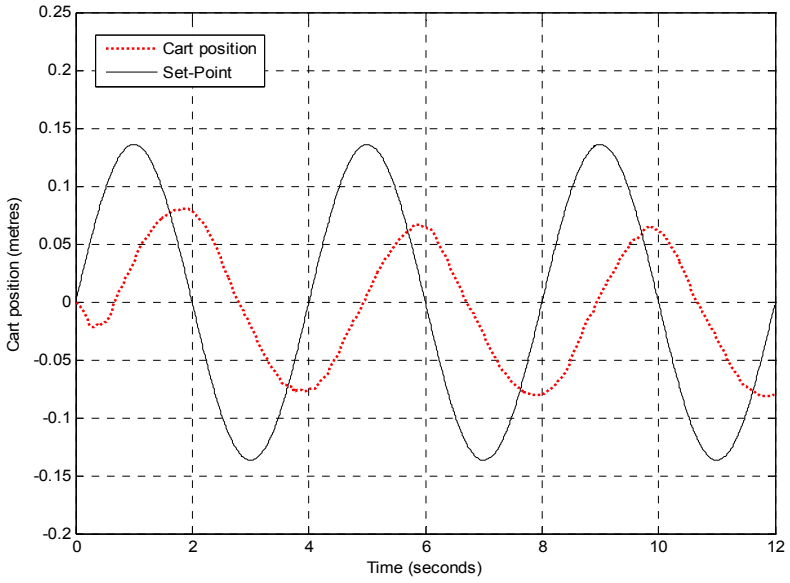

Figure 5-7: LQR Tracking control using a dynamical SP=0.135 sin (wt), Frequency=0.25Hz.

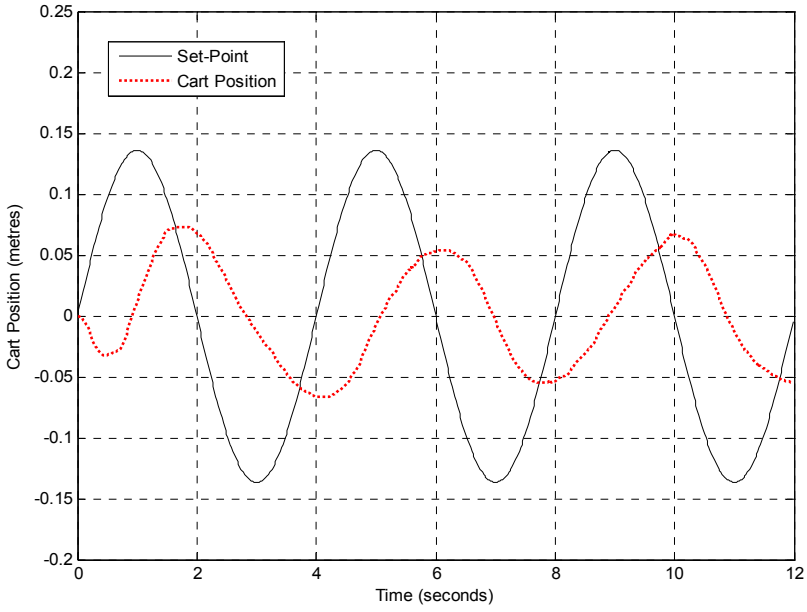

Figure 5-8: PID Tracking control using a dynamical SP=0.135 sin (wt), Frequency=0.25Hz.

The LQR controller used for the experiments shown in Figure 5-7, Figure 5-9 and Figure 5-11 gradually becomes more stable and follows the trajectory smoothly: that is, the errors reduce over time.

By contrast, the system under PID control becomes erratic and the error between the equilibrium point and the rod position does not improve over time (Figure 5-8, Figure 5-10, Figure 5-12 and Figure 5-14).

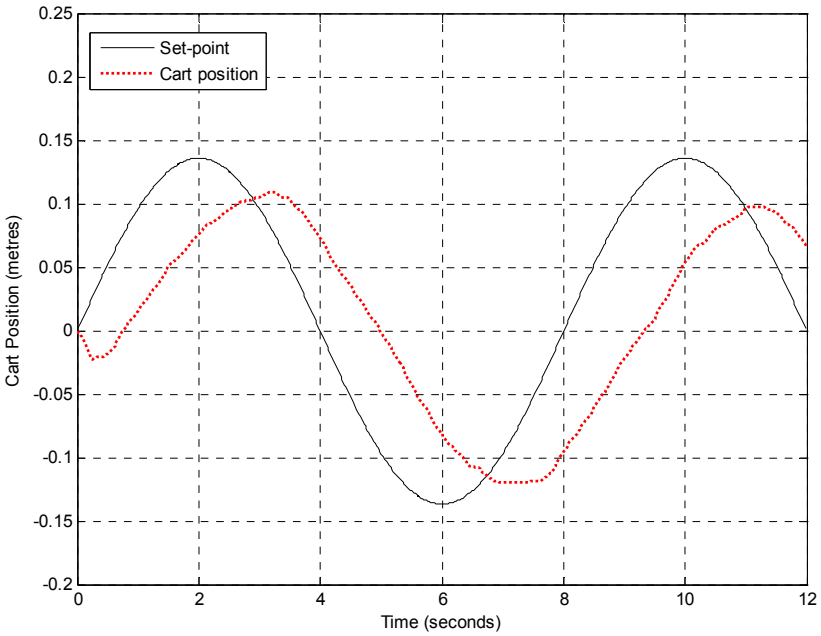

Figure 5-9: LQR Tracking control using a dynamical SP=0.135 sin (wt), Frequency=0.125Hz

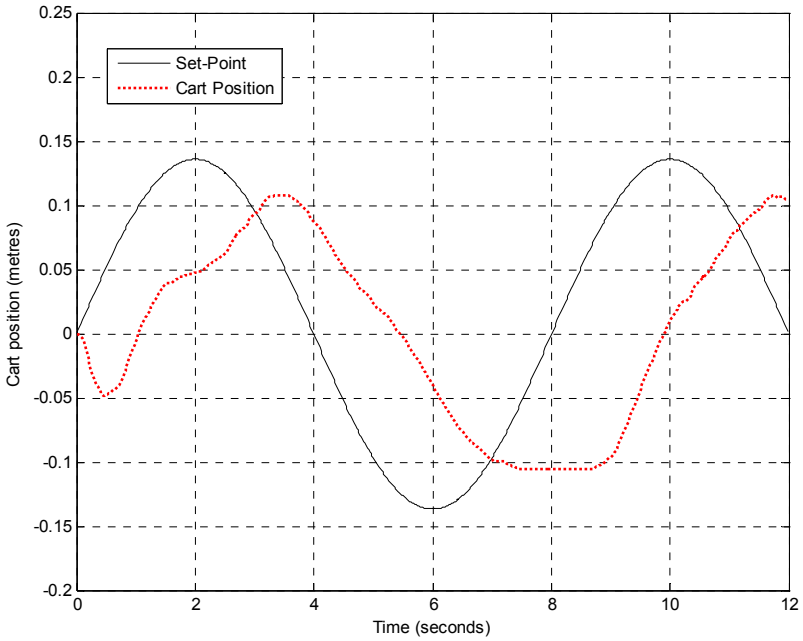

Figure 5-10: PID Tracking control using a dynamical SP=0.135 sin (wt), Frequency=0.125Hz

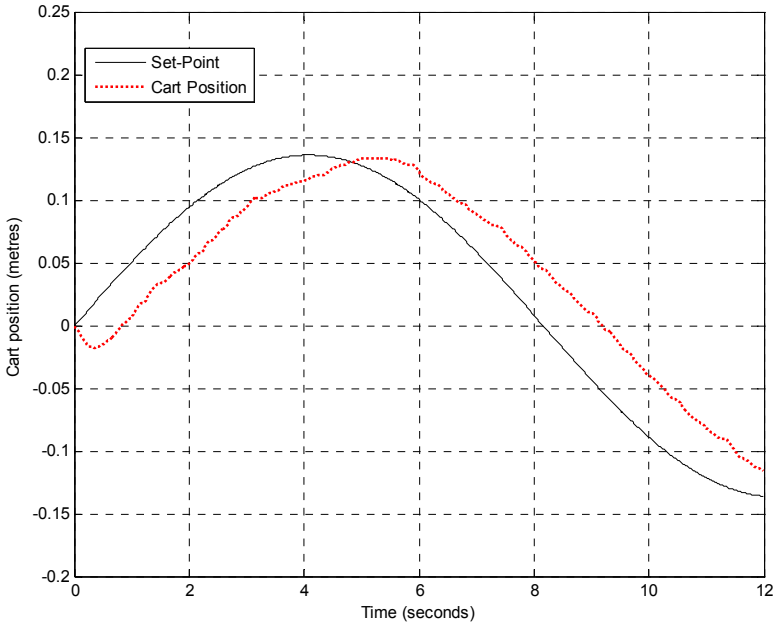

Figure 5-11: LQR Tracking control using a dynamical SP=0.135 sin (wt), Frequency=0.0625Hz.

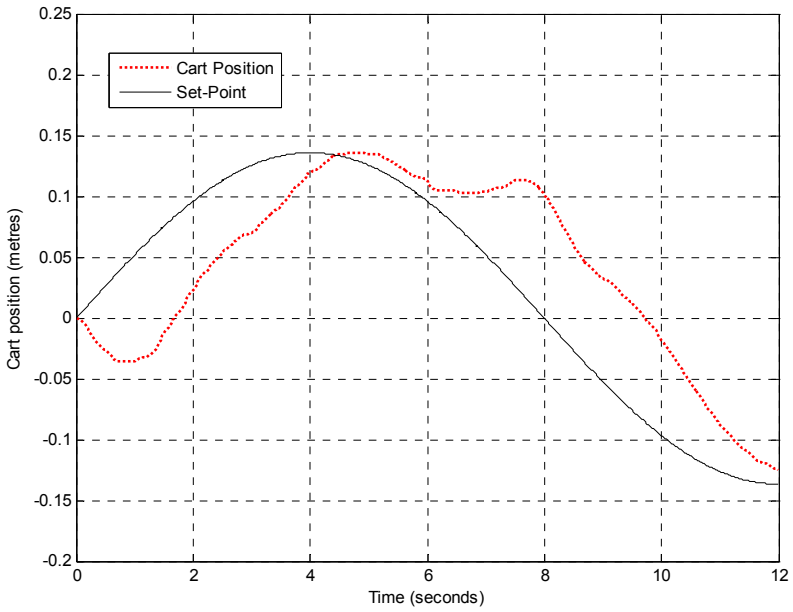

Figure 5-12: PID Tracking control using a dynamical SP=0.135 sin (wt), Frequency=0.0625Hz.

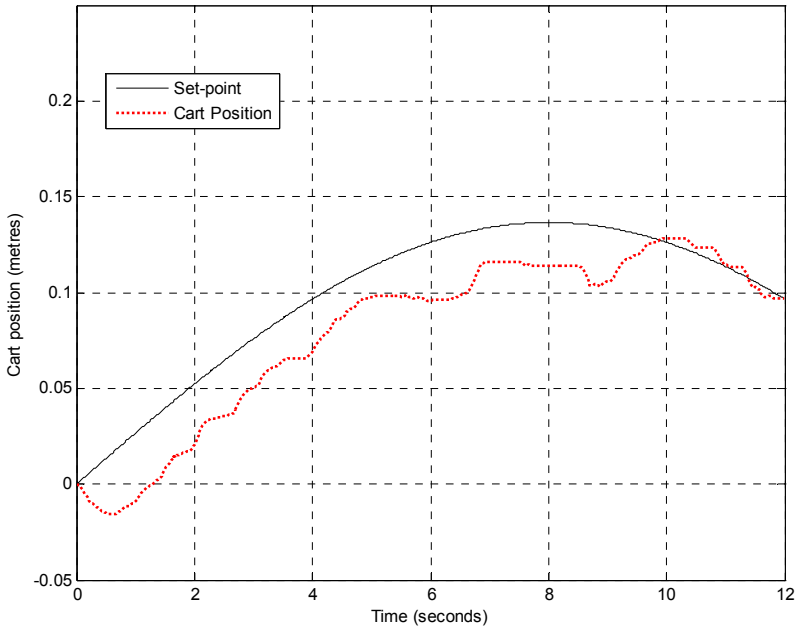

Figure 5-13: LQR Tracking control using a dynamical SP=0.135 sin (wt), Frequency=0.03125Hz.

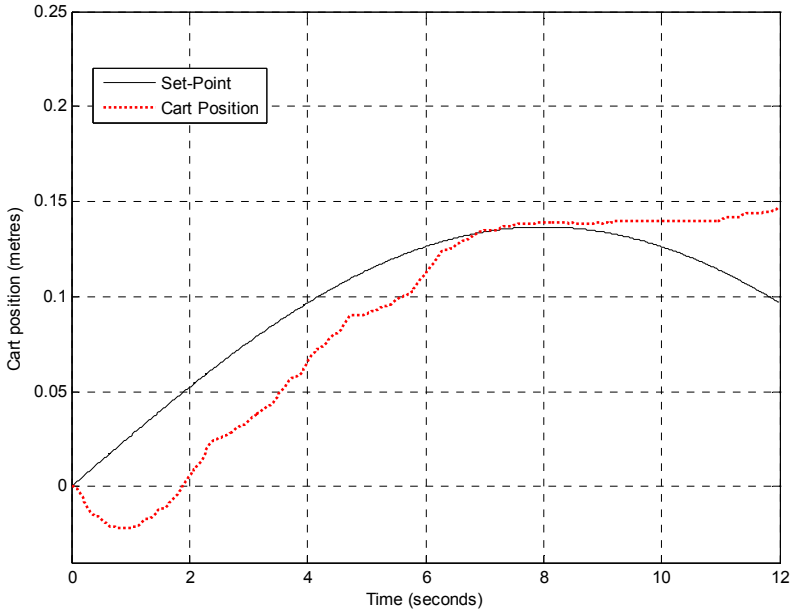

Figure 5-14: PID Tracking control using a dynamical SP=0.135 sin (wt), Frequency=0.03125 Hz.

5.7 Comparing the robustness of the two controllers

The robustness was tested experimentally in both system controllers. The mass and the length of the rod were changed without altering the gains of the algorithms.

Specifically, two different types of changes to the system were done:

- Adding a mass of 165 grams at the top of the rod
- Reducing half of the rod

Original system, SP=0:

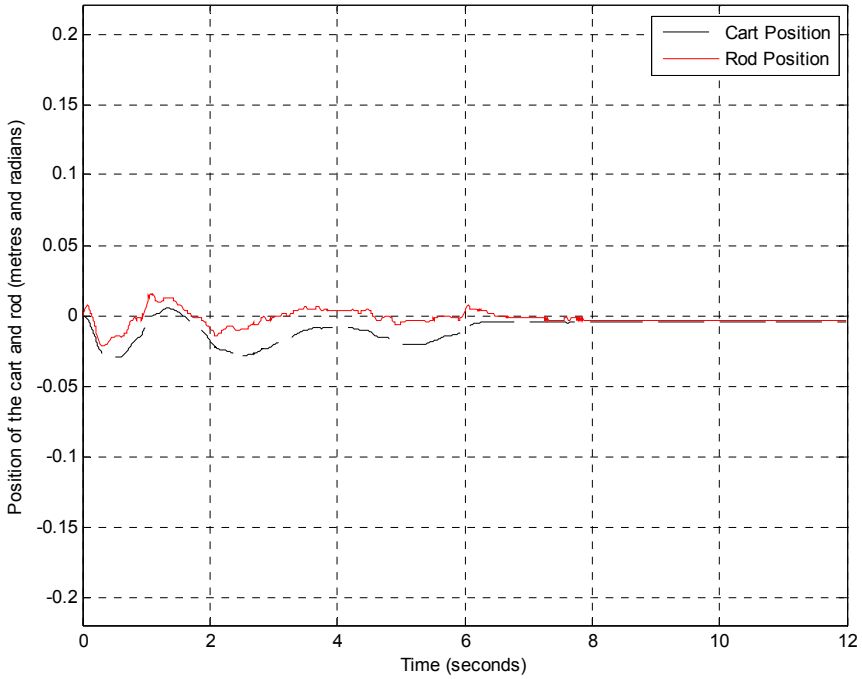

Figure 5-15: LQR controller, parameters fully known.

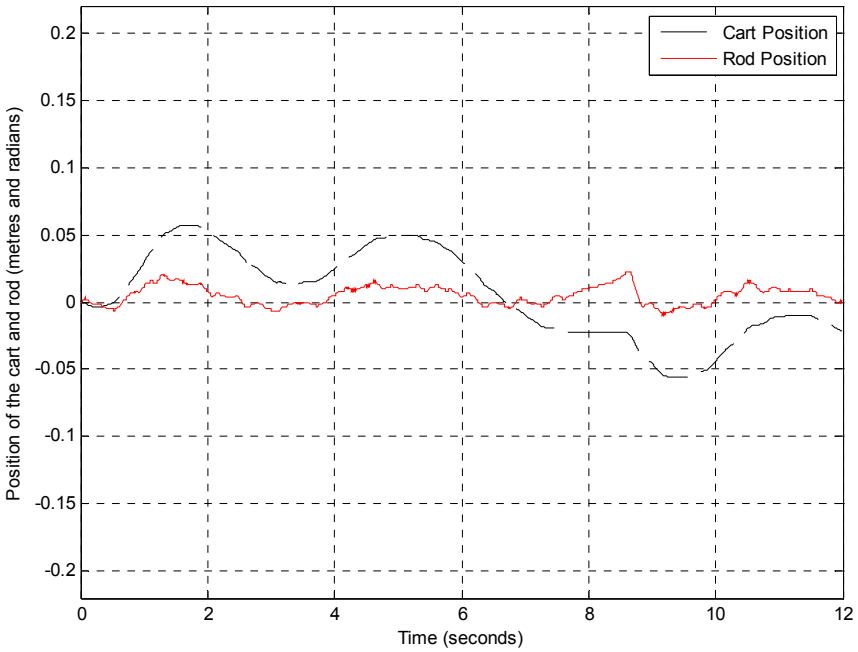

Figure 5-16: PID controller, parameters fully known.

In Figure 5-15 is shown the LQR system tuned to the original parameters: the system is stable and reaches the equilibrium point in around 8 seconds (for both states).

Figure 5-16 shows the behaviour of the PID controller. The gains used are the same as those used in the experiments shown in Figure 3-6. The system is stable; however, performance of PID is affected by external disturbances.

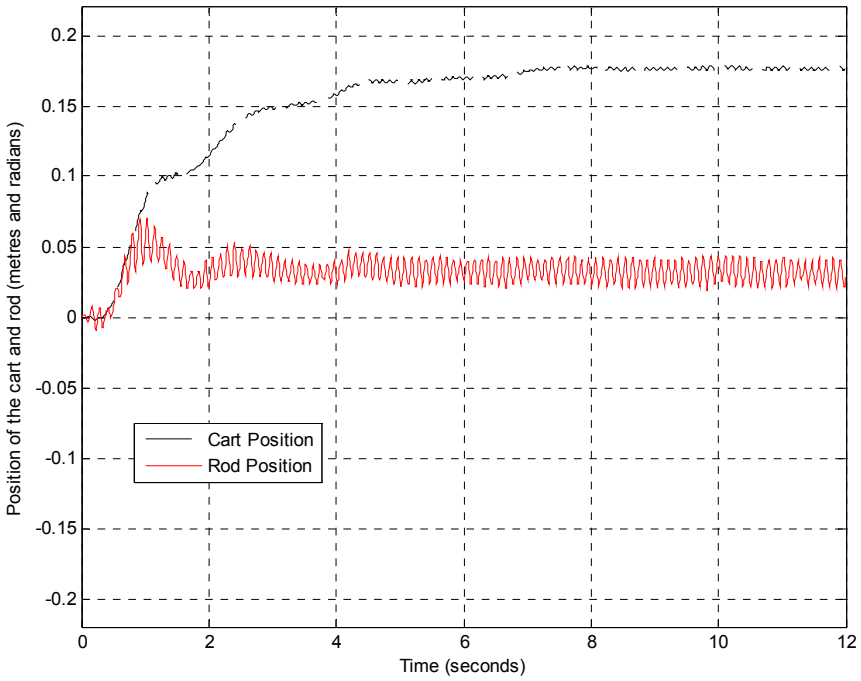

Figure 5-17: LQR controller, half-length of the rod.

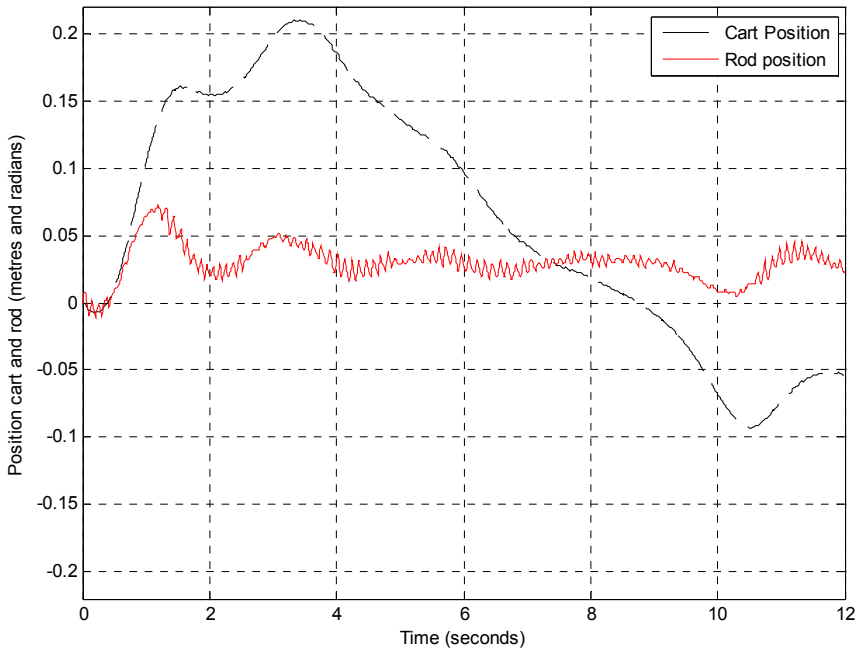

Figure 5-18: PID controller, half-length of the rod.

When the pendulum rod length is reduced by half, it can be seen that – for the LQR controller (Figure 5-17) - the offset of the cart position increases considerably and a measurable oscillation can be detected in the rod position. However the PID controller compensates this effect in the rod and shows better performance compared to the LQR see Figure 5-18.

Changing the rod mass:

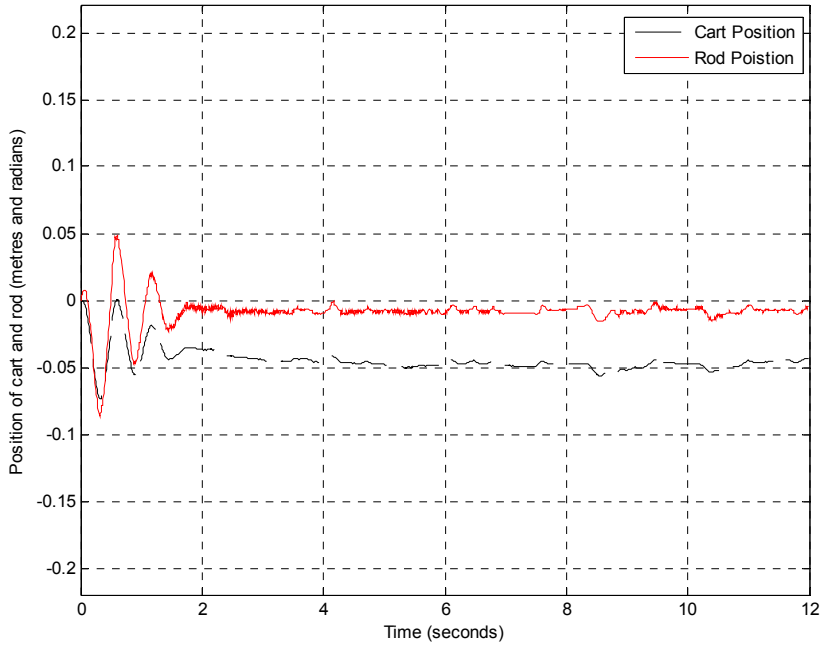

Figure 5-19: LQR controller, 165 grams in the top of the rod.

For the LQR controller (Figure 5-19), changing the mass of the rod has less impact than the change in the length: however, there is still oscillation in the rod position.

By contrast, the performance of the PID controller is more significantly degraded by the mass change (Figure 5-20).

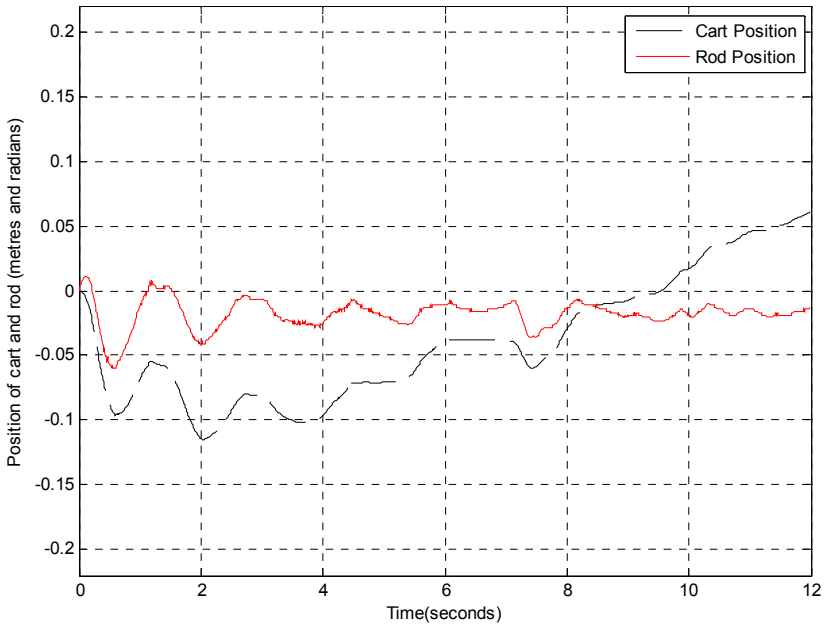

Figure 5-20: PID controller, 165 grams in the top of the rod.

5.8 Comparing the resources used for the two controllers

Although the way to obtain the gains for the LQR and the PID are completely different, the embedded microcontroller algorithms are essentially the same. Overall, there are just two important differences:

i) The PID controller has an integral term, which the LQR design does not include.

ii) Because of the lack of the integral term, the LQR algorithm does not include an anti-windup approach.

The number of additions and multiplications that perform the two control algorithms are the same; however, the compilation results show slight differences.

A brief discussion of these differences on the system resource requirements as follows.

5.8.1 Requirements for implementation of the two controllers

For the LQR controller, the following processes must be implemented:

i) The position information was acquired for the cart and rod using the microcontroller hardware (timers and interrupt handling).
ii) An approximation of the speed was obtained using pseudo derivates and digital filters.
iii) The control algorithm requires four states, position of cart and rod and their speeds.
iv) The program multiplies each state by a gain and adds the results. The total of these were four multiplications and four additions, after which the output is compared with a maximum value.
v) The program determines the direction of movement and uses this information to control the direction pin at the power stage.
vi) An offset is added to avoid the "dead zone" of the actuator (DC motor)

For the PID controller, the following processes must be implemented:

i) The position information of the cart and rod were obtained practically like the LQR program using the microcontroller hardware (timers and interrupt handling).
ii) An approximation of the speed using pseudo derivates and digital filters (different coefficients from LQR but the same filter order).
iii) The control algorithm requires five variables (position of cart and rod, speed of cart and rod, plus the integral of the cart position - including the stored integral).
iv) The program multiplies each state by a gain and adds the results. This amounts to five multiplications and four additions, after which the output is compared with a maximum value.

v) The program determines the direction of movement and uses this information to control the direction pin at the power stage.

vi) An offset is added to avoid the "dead zone" of the actuator (DC motor)

Table 5-3 Summary of the memory requirements for PID and LQR implementation

Algorithm	*Data*	*Constants*	*Code*
PID	1308	152	4328
LQR	1308	148	4064

Overall, according to Table 5-3 the PID has a slightly higher memory requirement than the LQR algorithm.

5.9 Conclusions

In the study described in this chapter, the control of an inverted pendulum system using both LQR- and PID-based embedded systems was considered and implemented.

Overall, the two systems have been shown (in the tests discussed here) to have similar control performance. In addition, the LQR control algorithm was seen to have similar resource requirements to the PID algorithm.

Chapter 6
H-infinity control for sensor-constrained mechatronic systems

This chapter[10] explores the robustness properties of H-infinity control for implementations with limited sensor resolution. The guidelines described in this chapter are the main contribution of this thesis.

6.1 Introduction

Based on limitations in the control algorithms shown in previous chapters, this section introduces a novel method that is intended to assist in the design and implementation of optimal H-infinity (H_∞) algorithms in low-cost mechatronic applications. This algorithm deals with uncertainties in the model plant and exogenous disturbances. The particular problem considered is position control in a situation where there are both sensor-related uncertainties (caused by low–resolution sensors) and limited computational resources. The first part of the method presented in this chapter describes how to design the H_∞ algorithm based on the dynamic features of the sensor. The second part of the method involves finding a suitable numerical controller representation in order to reduce memory and CPU load. Evaluation of the method is based on empirical studies using three industrial sensors employed in an under-actuated[11] robot. Results for a classic PID controller are included, in order to provide comparisons with the H_∞ approach.

---

[10] This chapter was presented in the paper: Bautista-Quintero, R. and Pont, M. J. "Implementation of H-infinity control algorithms for sensor-constrained mechatronic systems using low-cost microcontrollers" IEEE transactions on Industrial Informatics, Digital Object Identifier: 10.1109/TII.2008.2002703

[11] In the context of this chapter, "under-actuated" refers to mechanical systems that have fewer actuators than degrees of freedom.

6.2 Problems caused by low-resolution sensors

In low-cost systems, and in particular those with imperfect sensors, the quality of the sensors can lead to a degradation of the system's dynamic behaviour (because lack of sensor precision can be seen as uncertainty in the controller input signal [Bernstein, 2001]).

One feedback control algorithm that has the potential to overcome sensor-related uncertainties is the robust H_∞ technique [Zames, 1976], [Francis, 1987], [Gu et al., 2005], [Skogestad and Postlethwaite, 2005]. In spite of its theoretical maturity, the implementation of robust H_∞ presents some significant challenges when resource-constrained computer processors are employed [Chandrasekharan, 1996]. In addition, the design process deals with frequency-domain specifications that do not address the fundamental time-domain requirements directly [Selekwa, 2006].

In this context, this chapter introduces a method that is intended to assist in the design and implementation of effective H_∞ algorithms in low-cost mechatronic applications used for position control. The new method consists of two parts. The first part involves the derivation of numerical specifications for weighting functions using an H_∞ mixed-sensitivity approach that is based on the dynamics of the sensor. The second part involves finding an alternative numerical controller representation in order to implement the system using a microcontroller with limited resources.

Evaluation of the method is based on experimental studies using three industrial sensors of 1000, 2000 and 4000 pulses per revolution (ppr), employed in a sub-actuated robot (inverted pendulum). As in previous chapters, a classic PID is also used in order to provide comparisons with the dynamic H_∞ approach.

The remainder of the chapter is organised as follows. Section 2 is divided in two parts: the first part surveys previous work in sensor technology used for position control in mechatronic applications. The second part introduces fundamental issues of H_∞ control in order to illustrate the implementation challenges for resource-constrained computer systems. Section 3 explains how the dynamic features of the sensor can be used to define the weighting function of the controller and summarises the implementation method in detail. Section 4 presents the results from an empirical study that was used to evaluate the proposed method. Our conclusions are presented in Section 5.

6.3 Related work

In situations where there are few restrictions on memory and CPU resources, implementation of both classic and modern control algorithms will generally prove straightforward [Teng, 2000]. Even where the implementation places important limitations on computational resources, this is not necessarily a problem, provided that a simple control algorithm is used (e.g. see [Bautista-Quintero and Pont, 2006], [Henriksson, 2006] and [Palopoli, 2002]). However, where a dynamic control algorithm is employed in a complex multivariable problem, the implementation can present very significant challenges. Simplification of the program code or reductions in the size of variables are crucial issues if it is required to reduce resource requirements [Helton, 1995], but incorrect implementation decisions can cause the algorithm to fail [Raghunath and Parhi, 1994]. Additional challenges when implementing dynamic controllers in resource-constrained systems are related to sample-time selection [Åström and Wittenmark, 1984]; scheduler / operating system selection and implementation [Pont, 2001]; memory requirements [Grant, 1990]; impact of jitter [Proctor,

2001]; and issues related to the under-sensing[12] or under-actuating of the plant [Bernstein, 2001].

6.4 Selecting a cost-effective position sensor

This chapter is concerned with position-control applications. Many types of sensors are used for this purpose. For common industrial applications, examples include: displacement sensors used for pneumatic applications [Reininger, 2006]; low-cost potentiometers for position acquisition [Li and Meijer, 1998]; position estimation based on velocity measurements using high accuracy tachometers [Panda, 2003] and simultaneous sensing of position and speed using resolvers [Hanselman, 1989].

Of the available sensors, the optical encoder is the most widely used in long-life industrial applications [Ellis, 2004]. For example: this type of sensor is commonly used in servomotor control applications [Dote, 1990] (including recent high-accuracy systems [Kojima et al., 2004]).

In mechatronic control systems digital (optical) encoders are now suitable for use in high-reliability applications [Kojima et al., 2004]. These encoders are available in two basic forms: incremental [Eun-Chan, 2003] and absolute [Kulkarni, 2001]. In general – in terms of complexity, cost and efficiency – incremental encoders outperform the absolute type [Hebert, 1993]. For precision-control applications, the quality of the feedback signal depends fundamentally on the accuracy of the sensor. The resolution of digital encoders is limited by the slots through which the encoder light travels [Orlosky, 1996]. Advances in technology have tended to reduce the size of these slots and increase the bandwidth, in order to improve precision [DumbraÏ, 2000]. However, the trade-off between accuracy and cost is

[12] Under-sensing is the effect produced when a physical signal is quantified and the lack of precision (due to nonlinearities in the sensor or sensitivity to the environment), limits the range, resolution, linearity, etc. of the sensor signal.

unavoidable. In this context, various algorithms have been developed in order to increase accuracy while using the same sensor hardware. For instance the "*quadrature*" technique is an off-the-shelf technique which is used to increase the resolution of the sensor (by a factor of 4) in many industrial controllers such as programmable logic controllers (PLCs) and servo drivers [Orlosky, 1996].

Other methods that increase the resolution of incremental encoders are "clock pulse counting methods" and "analogue sine encoding" [Ellis, 2004]. One disadvantage of these techniques is that they require both additional timers and an analogue-to-digital converter. Alternative approaches have also been proposed ([Emura, 2000], [Tan, 2002]): these are based on interpolation and therefore require additional memory (for use in look-up tables). This can also add to costs.

Although a *quadrature* technique is a practical choice for a large range of mechatronic applications, the resolution enhancement provided by this technique may not be sufficient. Previous research in this field has considered this problem. For instance: high-performance control using low-resolution sensors was investigated by [Chang et al., 2001], however the results of this approach are useful only for speed control. Similarly, for industrial high-precision machines (*e.g.* numerically-controlled lathes), an algorithm based on an high-resolution interpolator was found to be an effective solution when sensor response was the main limitation [Emura, 2000]. Another effective technique for speed control of permanent-magnet motors was based on current and voltage measurements combined with the use of a low-resolution incremental encoder [Ogawa, 2001]. For position control a similar approach was followed using a low-resolution sensor plus a combination of an instantaneous speed observer algorithm and a sensor-less technique [Guidi, 1997].

The author of this thesis is not aware of previous studies that have focused on the specific design of robust controllers where the encoder resolution is limited.

Instead, the use of interpolators and look-up tables techniques (see [Hagiwara, 1992], [Kaul et al., 1997], [Takahashi and Wang, 2000] and more recently [Tan, 2002]) expensive high-resolution sensors have been the common solution for such implementation problems ([Magana and Holzapfel, 1998], [Kojima et al., 2004]).

6.5 Robust H_∞ control

Mathematical descriptions of physical models have played a crucial role in the history of modern feedback control [Lewis, 1992]; [Doyle et al., 1990]. Modern feedback control algorithms are evaluated using such models before being implememented [Gu et al., 2005]. Computer simulation provides strong evidence about the way in which the real system can be expected to behave in the field [Ellis, 2004]. However, due to the complexity of some dynamic systems [Doyle et al., 1990], incomplete or inaccurate models have often been employed. Even with imperfections, such models can still prove useful when developing feedback control laws, provided that the underlying control algorithms are "robust".

Seminal studies conducted by Isaac Horowitz and Patrick Rosenbaum in 1975 [Horowitz, 1975] provided the first formal notion of robustness for linear, time-invariant systems. George Zames later worked on the minimisation of the sensitivity function by feedback control schemes with respect to the H_∞ norm [Zames, 1981]. The mathematical representation for multivariable systems in state-space representation was developed a few years later by Doyle [Doyle, 1984]. After subsequent key contributions in this area [Doyle et al., 1989] and work on digital control [Åström and Wittenmark, 1984], the first compendium of this optimal approach was published [Zhou, 1996].

Despite the maturity of H_∞ control, this technique has not been used extensively for industrial applications [Gu et al., 2005]. Indeed, there is an apparent divorce between H_∞ theory and practical implementations. However, recent advances in software tools for designing H_∞ controllers (for instance, MATLAB and

Simnon[13]) and improvements in off-the-shelf hardware now make this algorithm a possibility for many industrial applications.

6.6 Optimal mixed-sensitivity design of H_∞ control algorithms

The goal in this chapter is to consider ways in which H_∞ control algorithms can be employed in low-cost embedded systems. In this section, a brief review of optimal mixed-sensitivity design of H_∞ algorithms is provided.

When seeking optimisation and increased robustness, an efficient modern technique is the optimal mixed-sensitivity approach, which in turn is based on the small-gain theorem.

The small-gain theorem states that if the loop gain is defined in an appropriate sense, such that is less than one, then the system is stable (See block diagram of Figure 1-1).

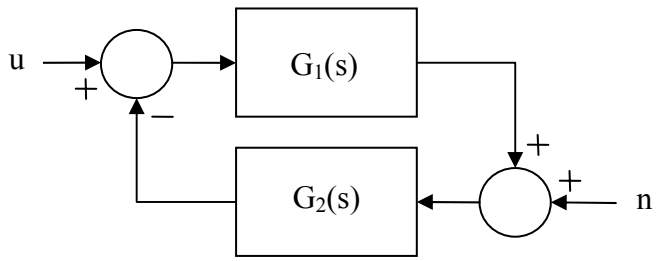

Figure 6-1 Block diagram whose the stability is studied using the small gain theorem

where *s* is the *Laplace* variable (*s= jw*), u the input 1 and n the input 2.

The capacity of *K(s)* to modify the system behaviour across a range of frequencies can be seen as the closed-loop shaping design. When the designer is able to

[13] http://www.sspa.se/software/simnon.html

modify the closed-loop frequency response, features like robustness can be defined directly (improving the gain and phase margins).

A general representation of a dynamic system (plant) and uncertainties in the model (disturbances, external measurement noise) is shown in Figure 6-2. The aim is that the controller *K* provides an output (u) such that the closed-loop system is kept stable in spite of disturbances (*d*), and external noise (*n*). The command signal or "set-point" (*r*) provides the desired value of the output *y*.

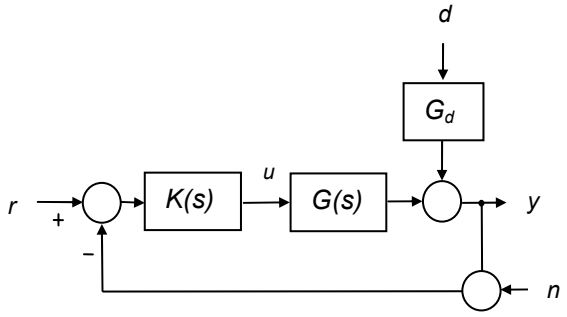

Figure 6-2 Block diagram of a feedback control system including disturbances (dynamic not modelled) and external noise (due to sensor imperfections).

The output *y* depends on three different sources: set-point, disturbances and noise. Using the superposition theorem, the output with regard to *r* when *n=0* and *d=0* is: $y_1 = (I + GK)^{-1} GKr$ Similarly considering *d*: when *r=0* and *n=0,* $y_2 = (I + GK)^{-1} G_d d$. Finally, considering *n*; when *r=0* and *d=0*; $y_3 = (I + GK)^{-1} GKn$.

Equation 6-1 shows the addition of the three effects together.

$$y = Tr + SG_d d + Tn \qquad \text{Equation 6-1}$$

where: $T = (I + GK)^{-1} KG$ which is the closed-loop transfer function, and $S = (I + GK)^{-1}$ is the sensitivity function.

The Bode plot of function *S* shows how the system is "sensitive" to disturbances in the frequency domain [Bernstein, 2000]. In a typical mechatronic system model, the magnitude of the sensitivity function is small for low frequencies and large for higher frequencies, contrary to function T. In fact $S+T = I$; so that in this perspective T is considered the complementary sensitivity function.

The objective of the mixed-sensitivity approach is to find a controller *K* such that the closed-loop gain must be kept small (from the small gain theorem) and several objectives must be satisfied simultaneously. These multiple objectives define the shape of *S* and *T* in the frequency domain in order to improve the robustness properties. A mechanism to provide the required shape of these functions is based on weighting functions. These weighting functions (filters in general) provide the desirable shape of *S*; *T* and *KS*, see Equation 6-2.

$$N = \begin{bmatrix} w_P S \\ w_T T \\ w_U KS \end{bmatrix} \qquad \text{Equation 6-2}$$

Weighting values are selected in conjunction with minimisation of the H_∞ norm of the closed-loop function (for instance in a second order SISO system this norm is the maximum amplitude of the output signal in the frequency domain, referred to as the "resonance peek"): see Equation 6-3. By finding the minimal value of this H_∞ norm of this expression, *K* becomes an optimal controller that keeps the closed-loop plant stable and robust (according to the specification given in the weights provided). For further details, see [Francis, 1987] and [Skogestad and Postlethwaite, 2005] .

$$\min_K \left\| N(K) \right\|_\infty \qquad \text{Equation 6-3}$$

In contrast to classical approaches (e.g. PID control), the designer of H_∞ has the ability to specify directly the robustness margins (and indirectly temporal performance), even for complex multivariable systems.

6.7 Designing the controller

As summarised in Section 6.1 the aim in this Chapter is to present a method that can be used to simplify the process of implementing a robust controller (based on an H_∞ algorithm) using a resource-constrained embedded processor.

6.8 Effective weight selection for sensor noise rejection

In the synthesis of the mixed-sensitivity approach, selection of appropriate weight functions provides the required behaviour of the closed-loop functions (such as T, S and KS). Based on this selection the controller is more or less able to tolerate uncertainties.

In order to select appropriate weight functions for the noise in the plant output (sensor signals) a simple taxonomy of this problem is presented. The literature divides such uncertainties into two types: 1) Structured, when uncertainties depend on the parameters of the plant (the model is not properly characterised, or the parameters change with the environment) and 2) Unstructured, when uncertainties are caused by dynamics which have not been modelled. For further details, please refer to Chapter 4 in [Doyle et al., 1990].

This chapter describes uncertainties caused by a lack of sensor-resolution which are categorised as unstructured. This type of uncertainty can be modelled as a multiplicative uncertainty of the plant output, see Figure 6-3 (please refer to [Francis, 1987] and [Gu et al., 2005] for further information).

Multiplicative uncertainty of the plant output is caused by: 1) Finite sensor resolution; 2) Phase lagging in the data-acquisition process; 3) Additional high-frequency dynamics which have not been modelled.

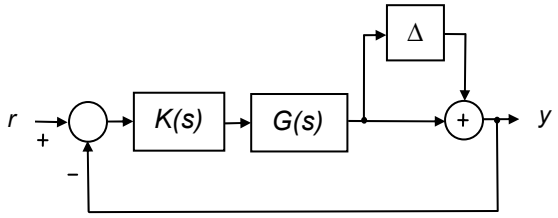

Figure 6-3 Block representation of multiplicative uncertainty of the plant output. For this application uncertainty Δ increases in proportion to the frequency of the encoder signal and in inverse proportion to the encoder resolution.

This kind of multiplicative uncertainty degrades the fidelity of the feedback signal, resulting in limited performance or even instability. To address this problem, the closed-loop function T is shaped in such way that a small gain is maintained at high frequencies in order to compensate for uncertainties in this range.

There are methods available which assist in the choice of weights in order to meet the frequency specifications [Papageorgiou, 1997], [Postlethwaite et al., 1990]. However using these methods, the need to meet both frequency and time domain requirements simultaneously is a trial-and-error process [Selekwa, 2006] As an alternative, it is proposed a systematic approach is proposed which can be used to determine the parameters of the weight functions based on the features of the system (sensor and microcontroller).

In this approach, the parameters of the weight function T are obtained by considering two fundamental constraints: 1) Sensor resolution and 2) computational resources. From these limitations, estimation of the frequency threshold at which uncertainties are large enough to affect the system stability is still a challenge. The method presented allows the weight w_T (see Equation 6-2)

to be set in such a manner that the designer can adjust the noise rejection, bandwidth and accuracy in the position control, in order to meet the requirements. Simultaneously (using w_P and w_U) this multi-objective approach allows the designer to trade-off features of the closed-loop functions: S and KS, related to performance and actuator boundary protection.

The minimal sensor(s) resolution in which the industrial mechatronic system can be implemented is defined according to the mechanical configuration of the attached sensor (see Table 6-1).

Table 6-1 Relation between the mechanical configuration and type of encoder selected (rotational, linear) in order to find the minimal static resolution (precision) reached for the end-effector.

Type of movement	***Mechanical configuration of the system:***	***Minimum resolution obtained (Δx) based on the sensor resolution available (either linear_res or angular_res)***
Linear movements	Measured by linear sensor	$\Delta x = linear\_res$ *Example: if Δx desired 0.1 mm* $linear\_res \le 0.1mm$
Linear movements	Measured by Rotational sensor	*There are three common cases:* 1. *Using ball-linear bearing* $\Delta x_1 = \frac{linear\_pitch\_mech\_bearing}{2\pi / angular\_res}$ 2. *Using pulley and timing-belt* $\Delta x_2 = \frac{\pi \times pulley\_diameter}{Gear\_ratio \times angular\_res}$ 3. *Using gear and linear gear* $\Delta x_3 = \frac{\pi \times gear\_diameter}{angular\_res}$
Rotational movements	*Measured by rotational sensor*	*Resolution of several joints depend upon the kinematics (geometry of the movement),* *For each single axis is:* $\Delta\rho = l \times \frac{angular\_res}{gear\_ratio}$ *For angular resolution:* $\Delta\theta = \frac{angular\_res}{gear\_ratio}$

where l is the length of the link coupled to the shaft of the rotational movement

Once the static resolution requirement is defined, a model that represents the position signal from the perspective of the embedded processor can be proposed: see Equation 6-4, where φ represent the signal obtained with the "ideal model" (without disturbances). The constant ρ represent the resolution of the system in pulses per revolution. The symbol $\lfloor \ \rfloor$ represents a rounding operation towards the closest integer (it is used to represent the discrete pulses counted from the digital encoder).

$$\hat{\varphi} = \frac{1}{\rho}\left\lfloor (z^{-1}\varphi)\rho \right\rfloor \qquad \text{Equation 6-4}$$

In the worst-case scenario the signal lags one sample period (z^{-1}). Longer than this period of time the scheduler loses synchronisation of the control algorithm.

The difference between the ideal and real signals increases: 1) in direct proportion to the frequency and 2) in inverse proportion to the resolution of the sensor.

Parameters of the weight function w_T can be obtained using the absolute difference between φ (using the numerical simulation of a pure signal with appropriate magnitude) and $\hat{\varphi}$ (using Equation 6-4). The frequency response of $e_r = |\varphi - \hat{\varphi}|$ is numerically estimated (see case study). The frequency at which the magnitude of e_r is greater than the precision requirements determines the cut-off frequency of w_T.

6.9 Optimising the controller for limited computational resources

It has been shown that the H_∞ mixed-sensitivity strategy is effective in terms of robustness even for multivariable systems. However the main disadvantage in

terms of implementation is the memory and CPU requirements compared with other classic control techniques [Henriksson, 2006].

Figure 6-4 illustrates the implementation requirements for this robust approach and the level of abstraction needed to create and implement such an algorithm. Note that the highest level of design starts with a model representation of the system. The "time-triggered co-operative" (TTC) scheduler implementation (on the right of Figure 6-4) has been shown to demonstrate predictable behaviour in a range of challenging applications, see for instance: [Pont and Ong, 2002]; [Pont and Banner, 2002]; [Key et al., 2003]; [Hughes et al., 2005]; [Phatrapornnant and Pont, 2006].

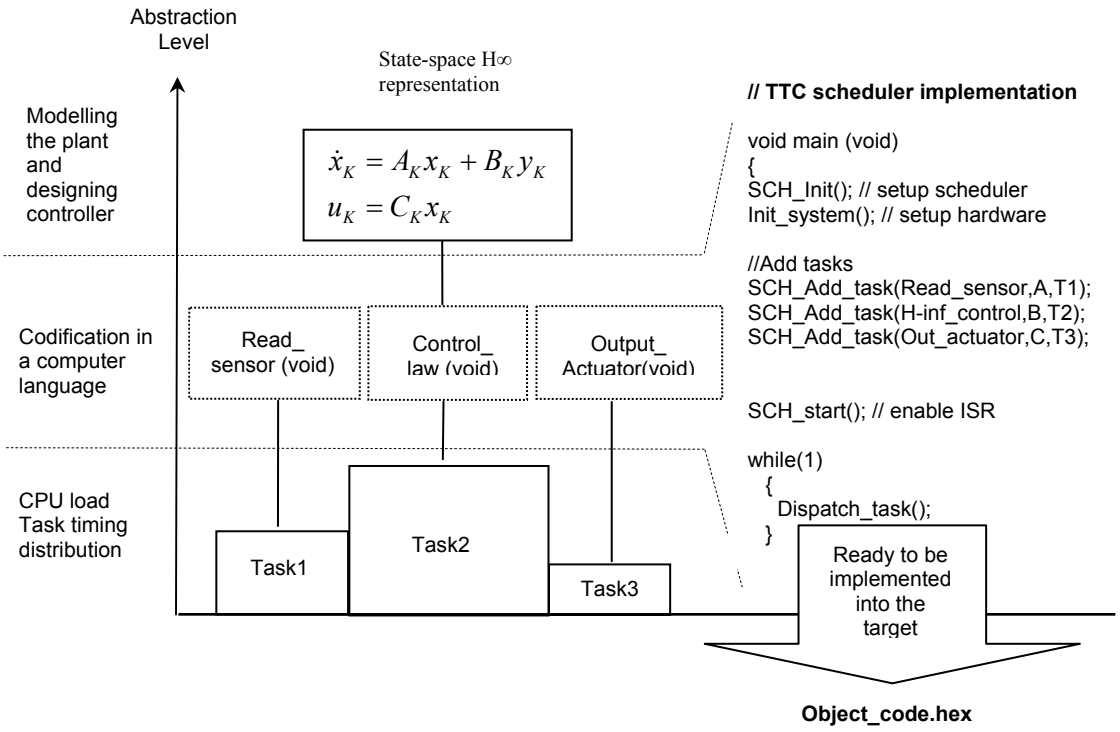

Figure 6-4 Basic representation of the design-implementation process of the H_∞ algorithm using a TTC scheduler. From the segment program A, B, C are the offsets and T1, T2 and T3 are the number of ticks: for further details, see [Pont, 2001].

The synthesis of the controller is obtained using numerical tools based on the weight functions and model provided. The controller has a state-space representation in order to allow direct implementation in a digital controller. Dimensions of the matrices and vectors are: $x_K \in \Re^{nx1}$; $A_K \in \Re^{nxn}$; $B_K \in \Re^{nxm}$; $u_K \in \Re^{mx1}$; $C_K \in \Re^{pxn}$; $Y_K \in \Re^{px1}$;

where:

n is defined as the number of internal states of the H_∞ controller. $n = (plant\_order) + (weights\_order)(number\_plant\_outputs)$

m is defined as the number of inputs to the H_∞ controller. This is equivalent to the number of plant outputs (sensors)

p is defined as the number of outputs to the H_∞ controller. This is equivalent to the number of plant inputs (actuators).

The control algorithm requires that values are read from the plant sensor(s) in order to calculate the value of y_K. This numerical value is used to adjust the actuator(s). The core of the control code (not including scheduler, data acquisition and actuation components) requires *Mr* memory units (Equation 6-5).

$$Mr = (m + p + n)(n + 1) \qquad \text{Equation 6-5}$$

As can be seen from Equation 6-5, *Mr* depends on the plant order, number of inputs and outputs of the plant and weights inclusion. The size memory required depends upon the precision of the numerical values of A_K, B_K, and C_K used in the synthesis of the H_∞ controller.

It is important to notice that mixed-sensitivity synthesis provides a state-space realisation might require high-precision variables for numerical calculations. The

high level of precision is required because of the large dynamic range of the C_K and B_K and also numerical condition.

Similar to Theorem 4 discussed in [Laub et al., 1987], an evaluation of the upper and lower bound of matrix C_K and B_K is obtained. This is used to estimate the range of numerical excursion of the matrix C_K (the synthesis of H_∞ provides large numerical values) and matrix B_K (the synthesis of H_∞ provides small numerical values). The ratio of the absolute maximum value of C_K over the absolute minimum value of B_K is calculated and compared with the precision of the program-variable size used. If the controller representation is poorly balanced (it is considered balanced if $C \approx B$), the precision of the variables has to be increased. This calculation is shown in Equation 6-6, where *num_bits* is the number of bits allocated for every single variable employed in the (control) program code.

$$Ratio\_CB = \frac{|\max(C_K)|}{|\min(B_K)|} < 2^{num\_bits} \qquad \text{Equation 6-6}$$

When the *Ratio_CB* exceeds 2^{num_bits} the precision of the variables should be increased. Consequently the total memory allocation for the core (Mr) grows in the same proportion. For embedded systems with limited resources, it is desirable to balance the numerical representation in order to make optimal use of resources. The balanced realisation process [Laub et al., 1987] finds an alternative to the state-space model of the controller such that *Ratio_CB* is reduced considerably.
This balance process is shown in Equation 6-7, the second state-space model is such that, TAT^{-1} is similar in numerical range to *TB*.

$$\dot{x} = Ax + bu$$
$$y = Cx + Du$$

Equation 6-7

$$\dot{\bar{x}} = TAT^{-1}\bar{x} + TBu$$
$$y = CT^{-1}\bar{x} + Du$$

Nevertheless there are other realisations used to simplify operations in the implementation stage. For instance Jordan forms and other canonical realizations can be implemented to reduce the number of operations performed in the microcontroller. These representations are studied in [Chen, 1999].

Similarly Delta-operator base representations (deal with balance realizations) can offer important advantages when the implementation is performed using a fix-point arithmetic processors.

The case study will show quantitatively the memory reduction that can be obtained using only the algorithm detail in [Laub et al., 1987].

The following guideline summarises the proposed implementation process:

1. In order to obtain the static precision required for the application, choose a sensor resolution ρ which is appropriate for the mechanical system configuration (see example in Table 6-1).
2. Find the maximum frequency (w_{max}) that meets the dynamic precision requirements. To do this, obtain discrete numerical values of the "ideal" sensor signal $\varphi = A_{\max} \sin(wT)$ considering 3 aspects:
   - Appropriate A_{max} according the span of the real variable.
   - Find an appropriate range of frequency, according to the sensor frequency response (given by the manufacturer).

- Select sample time (Ts) according to the plant dynamics. Based on the previous numerical values the signal $\hat{\varphi}$ is calculated using Equation 6-4. Plot the absolute error between signal models along a range of frequency ($e_{\varphi} = |\varphi - \hat{\varphi}|$). A suitable value of w_{max} will be found in the plot where the error e_{φ} is smaller than the precision required.

3. Weight functions W_T and W_P can be considered as first-order filters; in that case, there are three values to consider in the design: the bandwidth (w_B), the bound at high frequency (M) and the bound factor at low frequency (A).

4. W_T^{-1} gives the shape to the closed-loop function T, having: $W_T = (s + w_B / M)/(A \cdot s + w_B)$, so that, w_B is proposed with the next criterion: $w_B < w_{max}$, having M>1 and A<1.

5. W_P^{-1} gives the shape of the sensitivity function S. The filter developed can be $W_P = (s/M + w_B)/(s + A)$, see the trade-off between tracking and disturbance rejection responses shown in [Skogestad and Postlethwaite, 2005].

6. Based on weights W_T , W_P and W_U and the plant model use the standard MATLAB function *mixsyn* (or equivalent) to obtain the controller (MATLAB function *ssdata* can also be used to obtain the state-space representation for implementation). Note that weight *Wu* restricts the output signal (actuator), so that, *Wu* can be a constant in order to meet the actuator restrictions. See program code examples in [Skogestad and Postlethwaite, 2005].

7. Verify that the controller and plant in closed-loop maintain poles in the left half plane using the MATLAB function *eig* (or equivalent), to provide the eigenvalues of the closed-loop system. If there are right-hand plane poles, weight filters must be changed either by reducing the bandwidth w_B or increasing A.

8. Once the system is stable in simulation, obtain the space-state representation of the controller and find a balanced realization using the MATLAB function *balreal* (or equivalent): see [Laub et al., 1987].
9. Find the equivalent controller in the discrete domain. If the processor cannot meet the sample-time requirements, return to Step 2 and try a longer sample interval. Note that there is a trade-off between processor speed and plant timing requirements.

6.10 The testbed

As noted in the introduction, the aim of the case study presented in this section is to describe the design and implementation of an optimal H_∞ controller. The plant used is an inverted pendulum and the controller is a low-cost ARM-based microcontroller. This testbed has a horizontal linear movement and a rotational movement.

As a benchmark, an equivalent PID-based controller is also developed. To allow a quantitative comparison of the different implementation strategies, the code size, memory requirements and execution time is presented for both the PID and H_∞ systems.

6.11 Processor hardware and software architecture

As in previous chapters, the microcontroller employed in the studies described in this section was an NXP LPC2129 device based on a 32-bit ARM7 core. The code was written in C. The states were represented using 32-bit floating point variables.

6.12 H_∞ implementation

In this subsection, an implementation of this technique is introduced for solving the particular problem of optimising the sensor signal in spite of limited resolution.

Following Step 1 the minimal linear displacement (cart) sensed is calculated using the expression of Table 6-1. The mechanical configuration used is: *"linear movements measured by rotational sensor"*. The minimal angular resolution is given directly by the sensor resolution since there is a direct mechanical coupling between sensor and rod (see expression in Table 6-1, *"rotational movement measured by rotational sensor"*, case *gear_ratio*=1).

Once these static requirements are satisfied, the next part deals with the dynamic response of such sensors. This procedure selects the weighting function W_T =diag[W_{Tcart} ; W_{Trod}] based on the dynamic response of the sensor. The sensor is simulated using three different sampling times (according to the plant dynamic) and 3 different sensor resolutions (off-the-shelf sensors). The span amplitudes of the numerical signals are given from the real test-bed (A_{1cart}, A_{2rod}). Although selection of both sensors is important, it will be shown that rotational movement defines stability and performance as it was seen in the previous PID approach (see Chapter 3).

The maximum absolute error values are plotted along a dynamic range of frequency (see Figure 6-5).

For this testbed the maximum frequency selected was w_{max}=8 Hz and Ts=10ms. The minimum sensor resolution is 1000 ppr. These values were chosen in a trade-off between resources of the microcontroller, sensors available and dynamic error allowed for the application.

The weighting function W_T parameters are obtained based on Step 2, Step 3 and Step 4 from Section 6.9 and the W_P parameters are chosen based on Step 5.

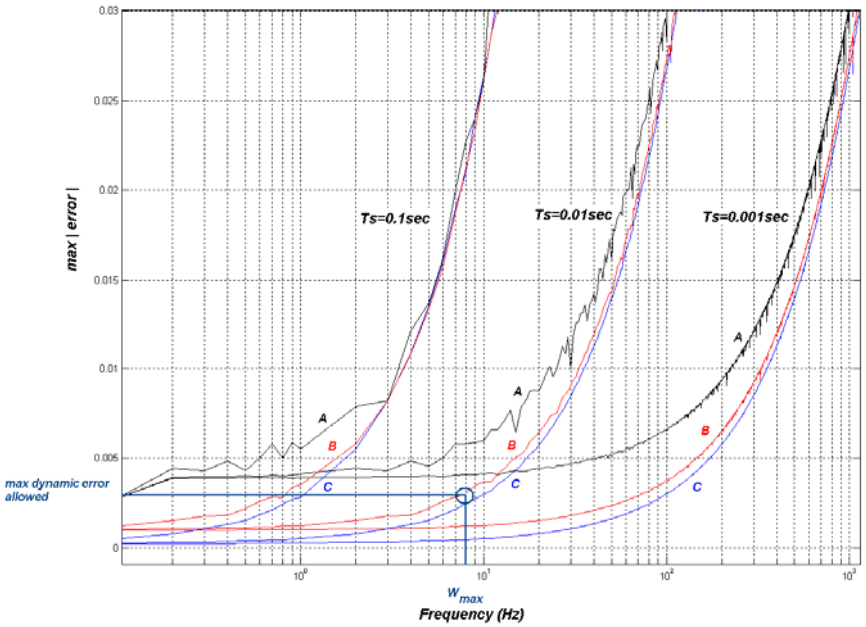

Figure 6-5 Frequency response of the maximum error between the estimated sensor signal φ and the disturbed signal $\hat{\varphi}$ using three sampling periods (Ts=0.1 s; Ts=0.01s; Ts=0.001s). A is the plot using a sensor of 256 ppr; B is a sensor of 1000 ppr and C is a sensor of 4000 ppr. w_{max} is approximately 8 Hz based on the maximum dynamic error.

Weight parameters can be adjusted manually if the optimal solution cannot meet all the requirements simultaneously. In this study *Wu* was left empty (b) in order that the algorithm executed by *mixsyn* could find an optimal solution based only on T and S closed-loop functions.

$$W_P = diag\left[\frac{\frac{s}{1.9}+1}{s+0.01};\ \frac{\frac{s}{1.9}+8}{s+0.08}\right] \qquad \text{Equation 6-8}$$

$$W_T = diag\left[\frac{s+\frac{1}{1.9}}{0.001s+1};\quad \frac{s+\frac{8}{1.9}}{0.001s+8}\right]$$

Based on Step 6 a general representation of the controller is obtained, by using the MATLAB function *khinf*. This representation can be translated to state-space representation using the *ssdata* command. This matrix representation is used to obtain the *Ratio_CB* using Equation 6-6 is possible. In order to satisfy this condition, 64-bit variables are required. However, using the balanced representation, *Ratio_CB* requires only 32-bit variables (see Table 6-2). The core of the control algorithm is not only efficient in terms of memory but also the CPU load is reduced by (at least) the same proportion.

Following Step 7 the poles of the system should be in LHP.

Table 6-2 Memory requirements for the H_∞ controller. Fourth order plant, 2 outputs (cart position and rod position), 1 input (actuation voltage) First order weighting functions (m=2 p=1 n=8 and Mr=99).

Controller	*Ratio_CB*	*num_bits (precision required)*	*Total bytes*
Khinf1	$1.829016253 \times 10^{15}$	64	972
Khinf2 (balanced)	$6.448374856 \times 10^{5}$	32	396

The real-time test of the system is shown in Figure 6-6. The numerical simulation includes the limitations in the sensor (4000 ppr). However, additional uncertainties (external disturbances, nonlinearities) can affect the behaviour in the time domain. When the system uses the 2000 ppr sensor, the system shows some degradation in the control performance. Nevertheless the stability is still achieved even when the lowest resolution sensor (1000 ppr) is employed (see Figure 6-7 and Figure 6-8).

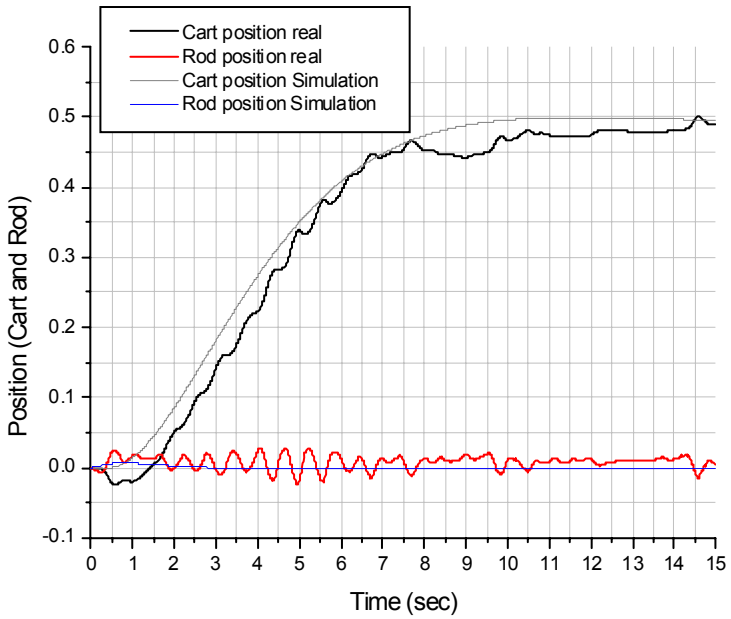

Figure 6-6 System using H_∞ with 4000 ppr rod sensor resolution, the set-point for the cart is 50 centimetres, and 0 radians for the rod angle. Positions in the real experiments show behaviour similar to the numerical simulation, however small differences are shown; they are caused by uncertainties in the model and external disturbances.

Memory requirements for the optimal H_∞ approach are considerably larger than those for the PID controller, however, both algorithms can fit in the same low-cost embedded processor. Similarly, while the H_∞ execution time is about ten times longer than PID, this still represents only 4% of the available processing time in this TTC architecture.

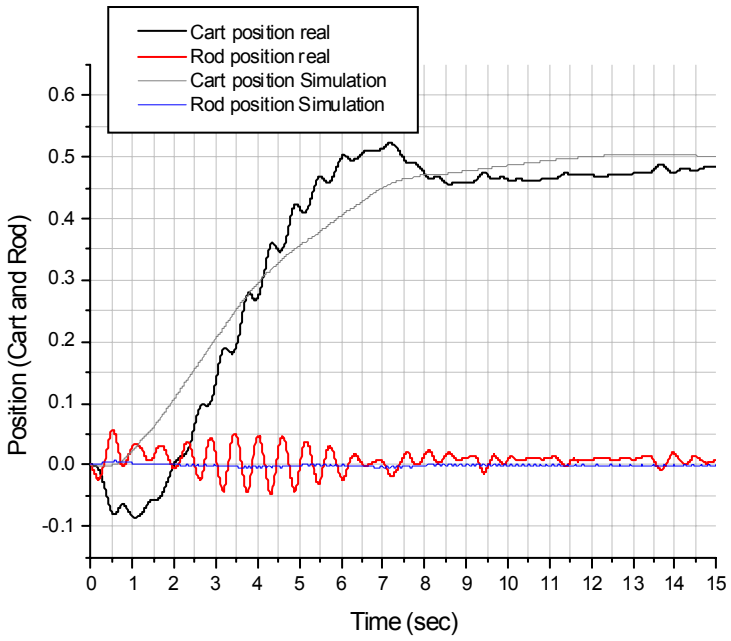

Figure 6-7 System using H_∞ with a rod sensor resolution of 2000 ppr. By reducing the rod resolution, the performance of both, the simulation and the real experiment deteriorates. However, the robustness of the mixed-sensitivity approach maintains a stable system.

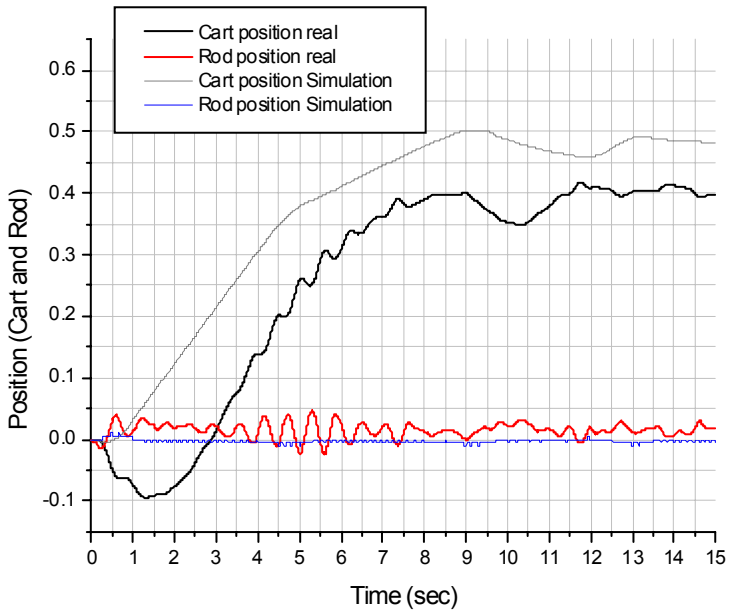

Figure 6-8 System using H_∞ with 1000 ppr in the rod sensor resolution. In contrast to the previous experiments, the reduction in the resolution of the rod sensor has an impact to the performance. However the system remains stable. In contrast, the PID algorithm was not robust enough to keep the system stable.

Memory and execution-time data for the two controllers are shown in Figure 6-9.

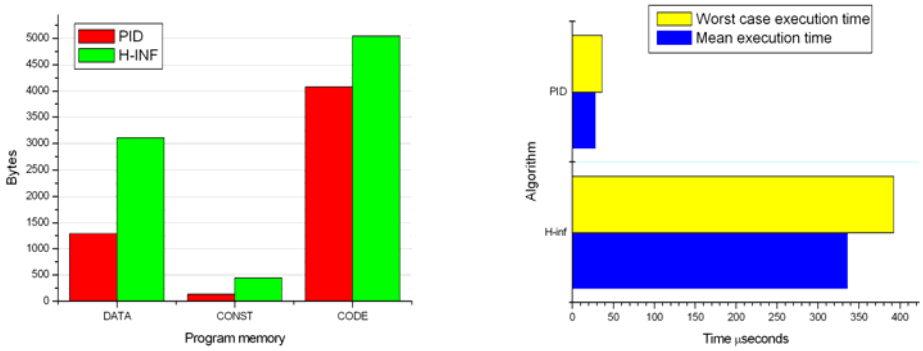

Figure 6-9 Comparison of memory and execution time between PID and H_∞ control algorithms.

6.13 Conclusions

For economic reasons, many control systems are implemented using simple sensors and actuators, along with small (embedded) processors which have (compared to "desktop" computers) very limited memory and CPU performance. Appropriate implementation strategies are required for such "resource constrained" systems.

This chapter has proposed a novel, practically-motivated approach for creating such system implementations. The method introduced in this chapter has two stages. The first stage involves the derivation of numerical specifications for weighting functions using an H_∞ mixed-sensitivity approach. The second stage involves finding an alternative numerical controller representation in order to implement the system using a microcontroller with limited resources. The creation of the numerical representation requires only the use of industry-standard toolsets.

Evaluation of the method was based on experimental tests using three industrial sensors with a non-trivial control problem (which is both non-linear and unstable in open loop). It was shown that the PID algorithm was of significantly lower

cost in terms of computational resources. It was also shown that both PID and H_∞ designs had similar levels of performance when the resolution of the sensor was relatively high (4000 ppr). However, when the sensor resolution was reduced, the PID controller was less able to tolerate disturbances, causing instability. By contrast, the H_∞ controller maintained the system stability even at the lowest levels of sensor resolution. For mass-production mechatronic systems, this optimisation may represent significant savings.

Chapter 7
Discussion

This chapter explains the meaning of the major findings presented in this thesis.

7.1 Introduction

The major finding in this thesis was the impact of what are considered non-conventional forms of feedback control using low-cost embedded systems in order to improve performance and robustness. During the course of the research, some practical techniques were integrated in order to facilitate implementation of control algorithms in a resource-constrained embedded microcontroller.

As outlined in Chapter 1, there is a clear gap of knowledge between control theory and practical implementation techniques. Multiple efforts have been made to bridge the gap between the two. This research investigated some of the reasons for this gap and consequences in the development of cost-effective products based on embedded systems.

Chapter 2 showed the evolution of the feedback control implementation in the context of embedded systems. The current limitations caused for an extraordinary fast commercial demand of embedded products will be discussed.

According to Chapter 3, the importance of considering the dynamics of actuators and sensors is first addressed in this thesis, particularly when the system requires a non-trivial control algorithm to reach stability and good performance. The principal impediment for a resource-constrained embedded processor was studied and discussion in this perspective is included.

The experimental validation presented in Chapter 4 shows the multiple impediments that make Fuzzy Logic Control (and related techniques) a challenging option for non-trivial control applications implemented using today's low-cost embedded systems.

In Chapter 5, the experimental results showed comparisons of two similar robust control techniques in terms of computational load. The two strategies were presented and the interpretation of the results clarifies the weakness and strength of each of them.

The exploration of limited resolution in the sensor capacity and the use of a robust but complex –in terms of computational burden– technique are discussed based on experiments presented in Chapter 6

7.2 Initial discussion based on the hypothesis proposed

Although the evidence of the gap between control theory and practice has been noted in multiple publications, the hypothesis presented in this thesis holds that in state-of-the-art applications this gap is less restrictive to the success in the implementation of control algorithms than in low-cost applications.

This argument has very important implications, since today's low-cost technology of embedded microcontrollers allows implementation of advanced techniques. The gap can be reduced for low-cost control applications by employing robust control techniques that are able to compensate for uncertainties in the plant, actuators and sensors. Nevertheless, some impediments have been observed to the efficient implementation of such techniques.

One of the principal impediments is the software support; this is due to the mismatch of the developments on embedded software and hardware. While the hardware has expanded in multiple families with different functionalities,

developers find it complicated to carry out an efficient integration with the software.

Additionally, for industrial and commercial control applications, the control algorithms are based on simple designs, that do not always provide optimal performance. The integration of more advanced forms of control was fundamentally limited by the conservatism of industry. Similarly, investment in large periods of research and development is impractical for short life-cycle products.

7.2.1 Findings of the literature review

The survey presented in the literature review was focused on the investigation of the historical evolution of ECSs and the methods used to implement control algorithms. The references were chosen for the tractability and relevance to the research project. The most relevant references in sporadic cases provided particular details of the implementation techniques. In contrast to this, more technical but less tractable references provided more information in the experimental context.

A trade-off between relevancy and tractability compound the literature review and as it was expected, researchers and developers have maintained incompatible scenarios to bridge the gap between the theory and practice. However, in the last few years, multiple initiatives are taking place to study the formalisation for ECSs design and implementation.

7.3 Discussion of the benchmark implementation

As it was presented in Chapter 3, the benchmark control (PID implementation) demonstrated some of the reasons of its success in many applications. The implication of using PID control in both situations: 1) without a mathematical

model and 2) with a simple computational implementation (in terms of resources), makes PID an excellent choice for low-cost embedded systems. On the contrary, for non-trivial plants models, there is an important observation to be made from the testbed implementation. In the model presented in Equation 3-11 the actuator is not considered. In contrast, Equation 3-13 represents the model of the actuator and plant in a single dynamic system.

The algorithm implementation for the system presented in Equation 3-26, the filter proposed in Equation 3-1 and tuning rules can be applied in both systems, Equation 3-11 (no actuator model included) and Equation 3-13 (actuator model included). However, the plant-actuator model represents more accurately the real system; numerical models can be studied off-line (in a computer simulation). Consequently, the system is optimised for a particular requirement. In contrast, some industrial applications that employ simple control computers commonly deal with trade-offs to satisfy multiple requirements simultaneously [Whidborne and Yang, 2002].

Modelling and characterisation of actuators may not be an easy task, especially for non-stable plants in open loop. Similarly commercial testbeds (like the one studied in Chapter 2, Section 2.4), the dynamics of sensors and actuators remains unknown for the user. In classic control literature, the omission of such dynamics was encountered (perhaps for didactic purposes), this widens the theory-practice gap.

7.4 Is FLC a practical choice for ECSs?

Chapter 4 presented two different forms of FLC, in both cases the results given by the experiments are constrained in several ways. This section summarises these findings.

7.4.1 Implications of the empirical FLC design

When a control algorithm based on Fuzzy Logic is programmed in low computational resources, multiple trade-offs have to be considered. For instance, Code 4-1 (fuzzification); Code 4-2 (evaluation and defuzzification) and Code 4-3 (rules) provide the basic framework to implement the FLC. A comparison between the simulation model and the real-time implementation (Figure 4-2) shows stability in both cases. However, performance of the real-time experiment shows how the system is affected by the external disturbances not considered in the model. This custom FLC is unable to guarantee predictable behaviour since the rules were chosen empirically (18 rules Code 4-3) and are only a subset of the complete model (625 rules).

The histogram shown in Figure 4-3 reveals how large the task timing of FLC is in comparison to the task timing of the PID approach, showed in Figure 3-7. Nearly 5% of the sample time is used to process the FLC algorithm (considering only the small set of rules). If the 625 rules were considered, the processor chosen (LPC2129) would not be able to perform the complete execution time in 10 ms.

Similarly, the PID outperforms the FLC with regard to the jitter comparisons, and in some applications, large jitter may cause unpredictable behaviour.

7.4.2 Requirements for the neuro-FLC

Although the neuro-FLC approach successfully alleviated the tuning problem, the main drawback is related to the rule explosion problem. In the example presented in Section 4.6.1 the system adjusts the FLC based on ANFIS technique. In this example, the rule explosion problem is addressed by distributed controllers. The results showed important advantages when the interaction dynamic can be estimated numerically. A comparison of these techniques can be seen in Figure 4-9 (without decoupling) and Figure 4-11 (using the decoupling technique).

Similar results were obtained in case study 1 (page 77). Further rule reduction was not required in this case since the system had only one input and two outputs. Memory and task timing requirements were relatively close to those needed by the PID approach. However, there were no apparent advantages in terms of robustness with respect to the PID.

In the case study presented in Section 4.8 the system distributed the FLC was split in two subsystems in order to cope with the rule explosion problem. As in the previous experiments, a PID trained the system and the performance is shown in Figure 4-18. The trained controller cannot deal with the uncertainties of the input channel for two fundamental reasons:

1) The training procedure is limited (FLC1 does not consider the dynamic interaction caused by rod and FLC2 does not consider dynamic interaction caused by cart)
2) Disturbances caused by external uncertainties cannot be included in the trained procedure (each experiment presents a unique set of disturbances).

7.5 Discussion of the optimal control

In order to increase performance based on optimal criterion, LQR was explored in Chapter 5. Discussion of the empirical comparison between LQR and the classic PID implementation follows.

7.5.1 Comparing results

Implementation of the LQR requires an accurate model of the plant. In addition, weights Q and r have to be chosen experimentally (as far as the author is aware, this is still an open problem). In Figure 5-2 the numerical simulation is shown relatively closer to the experimental data in comparison to the PID graphs presented earlier in Figure 3-6. It can be also be seen by comparing Table 5-1 and Table 5-2.

The frequency response of the system using the LQR presented higher bandwidth in comparison to the system under PID control. See amplitudes and phases of signals in Figure 5-3, to Figure 5-14. Nevertheless, the PID controller is able to eliminate a sub-harmonic effect existing in the cart position observed in Figure 5-5 and Figure 5-6. This is due to the integral compensator of the PID that even dynamically is able to reduce the effect caused for the dry friction located in the mechanical track system.

The robustness was tested experimentally by considering two different scenarios: reducing the rod to half its length, and adding a mass to the top of the rod

In the first experiment, Figure 5-17 shows how the LQR stabilises the system by changing the cart set-point position. This is because the controller gains were originally designed based on a model with a rod length of 0.303 m. The robustness of the controller allows stability, however performance is seriously affected. The static controller (LQR) finds the proper output by increasing the steady state error (about 0.18 m). In contrast, the PID controller compensates (the length uncertainty) dynamically using the integral action, see Figure 5-18.

For the second experiment, the LQR controller keeps the rod around the equilibrium point and the cart presents small deviations from the set-point (in comparison to the previous experiment) see Figure 5-19. As in the previous experiment, the integral action compensated for the uncertainties that affected the proper cart positioning, (see Figure 5-20).

7.6 Finding a cost-effective alternative with H_∞ control

As outlined in Chapter 3, it is desirable to find a mechanism to design a robust controller to compensate the uncertainties of the input control signals. For this problem, Chapter 6 presented a solution based on H_∞ control.

7.6.1 Meaning of the model uncertainties

For embedded systems with two fundamental constrains, computational resources (limited memory and CPU capacity) and sensor resolution, a model that includes both limitations is presented in Equation 6-4. This equation models the signal in the perspective seen by the real processor. When the resolution decreases, the error (difference between the "ideal" and the "real" signals) increases in inverse proportion. Likewise, when the sample period is reduced the error increases in direct proportion. The general effect is plotted in the frequency domain (see Figure 6-5) and the error increases exponentially with the frequency.

A trade-off between sensor resolution and embedded system computer capacity is required to evaluate an optimal solution, see Figure 6-5. There are three sample-times (for different processor capacities) and three sensor resolutions. A balance between maximum frequency selection and maximum error allowed for specific applications can be selected for this plot.

7.6.2 Comparison with previous experiments

The control algorithm is designed based on optimal H_∞ resulting in a more efficient method for compensating uncertainties in the input channel. Figure 6-6 with a rod sensor of 4000 ppr, Figure 6-7 with a rod sensor of 2000 ppr and Figure 6-8 with a rod sensor of 1000 remain stable (particularly with the lowest resolution) in contrast to the previous approaches (see PID experimental data in Figure 3-9 and Figure 3-10).

The scope of this approach was limited to analysis of the rod sensor resolution since the cart sensor in the mechanical configuration selected (see Table 6-1) did not constrain the stability of the plant under any of the previously approaches presented.

7.6.3 Comparison of computer requirements

Although finding a balanced numerical representation of the system controller (H_∞) is not a novel issue, an important implication for embedded systems can be seen in Table 6-2 since not only the memory reduction required to implement the system is achieved but also the CPU load is reduced considerably.

When the system based on H_∞ is compared with the classic PID in Figure 6-9, there are important differences in both, execution time and data used for each algorithm. PID control is simpler; therefore, the memory and processing time are smaller than H_∞. Nevertheless, the same microcontroller was used for both algorithms.

7.7 Conclusions

This Chapter has summarised and discussed the results presented in this thesis.

Chapter 8
Conclusions

This chapter concludes the thesis. It is divided into three parts. The first part shows the findings relate to the general aims proposed in the introduction. The second part deals with the relevance of the principal contributions of the thesis. The third part presents specific recommendations for further research.

8.1 Introduction

The preliminary conclusion was integrated at the end of each chapter in order to provide an essential summary of isolated findings. Taken as a whole, this chapter concludes the thesis by summarising the important links between these findings.

The goals proposed and achievements reached are analysed in order to show any limitations of the research.

Contributions presented in this thesis suggest the following. First, the empirical and analytical tools of this work provided a simple framework (based on a Time-Triggered-Cooperative scheduler and modern control implementation techniques) to bridge the gap between control theory and resource-constrained embedded implementations. Second, it opened the discussion to include advance forms of control in common industrial control software in order to address the optimisation problem of multiple-objective requirements.

To conclude, further research is suggested in order to confirm the preliminary results of this research, as well as other forms of control not considered in this thesis.

8.2 Goals and achievements

This thesis suggested that by using "ideal" system components it might be possible to deal with simple theory. According to the literature review, this hypothesis is accurate; nevertheless, there were some applications (state-of-the-art components) where the control problem is still too complex to solve with traditional techniques.

When using low cost components in contrast, there are multiple hindrances to overcome in the implementation stage. For instance, Chapter 3 presented an implementation of the classic PID in a non-trivial control application. The low-cost actuator limits a straightforward implementation. The model of actuator and integration with the whole system dynamic were required.

The first part of the Chapter 4 dealt with supporting FLC applications. This chapter concluded with two important findings. First, when using the approach based on empirical rules (given by an expert); there is not a straightforward mechanism to adjust the membership functions. The dynamics of the system (where the resources are limited) cannot be easily represented using linguistic rules. The low cost components introduced nonlinear dynamics that is too complex to represent using a few simple rules. Second, according to the experiments explored using the combination of techniques (ANFIS and decoupling,) there is not evidence to simplify implementation of the proposed Neuro-FLC for resource-constrained embedded systems.

In order to improve the plant performance, LQR presented several advantages in terms of efficiency and robustness with regard to PID, FLC and Neuro-FLC. However, neither LQR nor any other previous controllers could cope with sensor uncertainties. Exploration of a robust technique to address this problem allowed implementation of a robust design.

Design and implementation of H_∞ alleviates the problem caused by low sensor resolution.

In conclusion, the main achievement of this thesis was the practical and theoretical method to integrate and implement a robust control technique for a non-trivial control problem using low-cost components.

8.3 Relevance of the contributions

During the course of this research project, some publications where presented in technical forums and accepted in different journals (see list of publications in page vii). This thesis is supported by these contributions along with additional research. The relevance of these publications and thesis is summarised in two parts.

First, according to the literature review chapter the topic of generating efficient control software (simple and reliable) for embedded products is gaining importance for economic reasons.

Second, the optimisation in resources for control applications (sensor, actuators and controllers) is beneficial for mass production systems (based on ECSs).

8.4 Future research recommendations

The scope of this thesis was focused initially to study techniques used in ECSs. Due to different circumstances, some advanced control techniques were explored. The approach based on H_∞ illustrated numerically and experimentally advantages in terms of robustness and performance. In the sensors selected for this approach (incremental encoders), resolution is a feature that determines precision in the feedback signal. Low resolution implies delays in the response and deformation of the ideal signal. Industrial incremental encoders used under normal operating

conditions do not undergo additional non-linear effects (such as dead zones and non-linearity). However, future work will evaluate the proposed techniques using different sensors which – in addition to low resolution – also include non-linearity in the signal response.

In addition, there are other forms of control and their implementation techniques in the context of low-cost embedded systems that worth to be explored. For instance, MPC has similar complexity in terms of memory and CPU requirements to FLC. Similarly, SMC appears as a suitable alternative to gain in robustness in some industrial applications. For low cost ECSs these two approaches can be considered in future research.

8.5 Conclusions

This Chapter concludes the thesis.

Bibliography

Ahson, S.I., Lamba, S.S., Chaube, M.C. and Chandra, N., 1983. A Microprocessor-Based Multiloop Process Controller. IEEE Transactions on Industrial Electronics, IE-30(1): 34-39.

Alvarez, G.Z., Kawakami, R., Galvao, H. and Takashi, Y., 1999. Extracting Fuzzy Control Rules From Experimental Human Operator Data. IEEE Transactions on Systems, Man and Cybernetics, 29(3): 398-406.

Apneseth, C., 2006. Embedded System Technology in ABB. Embedded System Technology.

Arteaga, M. and Kelly, R., 2004. Robot Control Without Velocity Measurements: New Theory and Experimental Results. IEEE Transactions on Robotics and Automation, 20(2): 297-308.

Åström, K.J., 1985. Process Control-Past, Present and Future. Control Systems Magazine, IEEE, 5(3): 3-10.

Åström, K.J. and Hägglund, T., 2001. The Future of PID. Control Engineering Practice, 9(11): 1163-1175.

Åström, K.J. and Wittenmark, B., 1984. Computer-Controlled Systems: Theory and Design. New Jersey: Prentice-Hall.

Atherton, D.P. and Majhi, S., 1999. Limitations of PID Controllers. In: IEEE (Editor), American Control Conference, pp. 3843 - 3847.

Audsley, N., Bate, L. and Crook-Dawkins, S., 2003. Automatic Code Generation For Airborne Systems. In: IEEE (Editor), Proceedings on Aerospace Conference, pp. 2863-2873.

Auslander, D.M., 1996. What Is Mechatronics? Transactions on Mechatronics, IEEE/ASME 1(1): 5-9.

Auslander, D.M., Takahashi, Y. and Tomizuka, M., 1978. Direct Digital Process Control: Practice and Algorithms For Microprocessor Application. Proceedings of The IEEE, 66(2): 199-208.

Backus, J., 1978. The History of FORTRAN I, II, and III Association of Computing Machinery 13(8).

Bahram, S. and Hassul, M., 1993. Control System Design Using MATLAB.

Balakrishnan, V., Su, Q. and Koh, C.-K., 2001. Efficient Balance-and-Truncate Model Reduction For Large Scale Systems, American Control Conference, pp. 4746 - 4751.

Balzert, H., 2000. Lehrbuch Der Softwaretechnik: Software Development. Spektrum Akademischer Verlag.

Baumann, G.V., 1967. An Electronic Fuel injection System For Automobiles. Bosch Technical Journal.

Bautista-Quintero, R. and Pont, M.J., 2006. Is Fuzzy Logic A Practical Choice in Resource-Constrained Embedded Control Systems Implemented Using General-Purpose Microcontrollers? in: IEEE (Editor), Proceedings of The

9th IEEE international Workshop on Advanced Motion Control, Istambul, Turkey, pp. 692-697.
Bautista-Quintero, R., Pont, M.J. and Edwards, T., 2005. Comparing The Performance and Resource Requirements of “PID” and "LQR" Algorithms When Used in A Practical Embedded Control System: A Pilot Study, Proceedings of The UK Embedded Forum, Birmingham.
Bekit, D.W., Seneviratne, L.D., Whidborne, J.F. and Althoefer, K., 1998. Fuzzy PID Tunning For Robot Manipulators.In: IEEE (Editor), Proceedings of The 24th Annual Conference of The IEEE, pp. 2452-2457.
Bellman, R., 1996. Adaptive Control Processes. Princenton University Press.
Bennett, S., 1993. Development of The PID Controller. Control Systems Magazine, IEEE, 13(6): 58 - 62, 64-5.
Bergin, T.J.J. and Gibson, R.G.J., 1996. History of Programming Languages II. Addison-Wesley.
Bernstein, D.S., 1998. Control Experiments and What I Learned From Them: A Personal Journey. IEEE Control Systems Magazine, 18(2): 81-88.
Bernstein, D.S., 1999. On Bridging The Theory/Practice Gap. IEEE Control System Magazine, 19: 64-70.
Bernstein, D.S., 2000. The Frequency Domain. IEEE Control Systems Magazine 20(2): 8-14.
Bernstein, D.S., 2001. A Plant Taxonomy For Designing Control Experiments. IEEE Control Systems Magazine, 21(3): 7-14.
Bernstein, D.S., 2002. What Makes Some Control Problems Hard? Control Systems Magazine, IEEE, 22(4): 8-19.
Betin, F., Pinchon, D. and Capolino, G.-A., 2000. Fuzzy Logic Applied To Speed Control of A Stepping Motor Drive. IEEE Transactions on industrial Electronics, 47(3): 610-622.
Bezine, H., Derbel, N. and Alimi, M.A., 2000. On The Reduction of Large Scale Fuzzy Controller, Proceedings of The international Conference on ACIDCA 2000, Monastir, Tunisia,, pp. 135–140.
Bishop, R.H., 2006. Mechatronics An introduction.
Bishop, R.H. and Dorf, R., 2004. Modern Control Systems. Prentice Hall.
Black, H.S., 1984. Stabilized Feed-Back Amplifiers. Proceedings of The IEEE, 72(6): 716 - 722
Bose, B.K., 1988. Technology Trends in Microcomputer Control of Electrical Machines. IEEE Transactions on industrial Electronics, 35(1): 160-177.
Bronzino, J.D., 1982. Computer Applications For Patient Care. Addison-Wesley Publishing Co., California.
Bruijn, P.M., Roodink, P.A.M. and Verbruggen, H.B., 1988. A Simple Implementation of Adaptive Predictive Controllers. In: IEEE (Editor), international Conference on Control pp. 517-522.
Butazzo, G., 2002. Hard Real-Time Computing Systems.
Butazzo, G., 2005. Rate Monotonic Vs. EDF: Judgment Day. Real-Time Systems, 29(1): 5-26

Campbell, M., 1999. Knowledge Discovery in Deep Blue. Communications of The ACM Vol. 42(No. 11).
Cervin, A., 2003. Integrated Control and Real-Time Scheduling Phd Thesis, Lund institute of Technology, Lund.
Clark, E., Forbes, J., Baker, E. and Hutcheson, D., 1999. Mission-Critical and Mission-Support Software: A Preliminary Maintenance Characterization. Crosstalk 12(6): 17-22.
Coelho, T., Liang, Y. and Valter, F.A., 2004. Decoupling of Dynamic Equations By Means of Adaptive Balancing of 2-Dof Open-Loop Mechanisms. Mechanism and Machine Theory 39(8): 871-881.
Colnaric, M., 1999. State of The Art Review Paper: Advances in Embedded Hard Real-Timesystems Design. In: IEEE (Editor), Proceedings of The IEEE international Symposium on industrial Electronics, pp. 37-42.
Cuatto, T. Et Al., 1998. A Case Study in Embedded System Design: An Engine Control Unit. In: IEEE (Editor), Proceedings Design Automation Conference, pp. 804-807.
Chandrasekharan, P.C., 1996. Robust Control of Linear Dynamical Systems. Academic Press.
Chang, T., Chung, W. and Cohen, E., 2001. Speed Control of Brushless Motor Using Low Resolution Sensor, American Control Conference, pp. 293 - 298
Charette, R.N., 2005. Why Software Fails. Spectrum, IEEE, 42(9): 42- 49.
Chen, C.-T., 1999. Linear Systems Theory and Design. Oxford Series in Electrical and Computer Engineering.
Chen, X., Pu, H., Wang, X., Sun, Y. and Jia, W., 2007. Control System of A Modular and Reconfigurable Multilegged Robot. In: IEEE (Editor), international Conference on Mechatronics and Automation, pp. 1926-1931.
Dahlgren, F., 2001. Future Mobile Phones -Complex Design Challenges From An Embeddedsystems Perspective. In: IEEE (Editor), Seventh IEEE international Conference on Engineering of Complex Computer Systems, pp. 92-94.
Dicheng, L., Qing, W., Jiangfeng, Z. and Zhi, L., 2004. The Study of Microcomputer Adaptive Control System of Controlled Rectifier Device. In: IEEE (Editor), The 4th international Power Electronics and Motion Control Conference, pp. 704-707.
Donaghey, L.F., 1976. Microcomputer Systems For Chemical Process Control. Proceedings of The IEEE, 64(6): 975-987.
Dongkun, S. Et Al., 2002. Energy-Monitoring Tool For Low-Power Embedded Programs. Design & Test of Computers, IEEE, 19(4): 7-17.
Dormoy, J.-L., 2005. Artemis Research Priorities. In: ARTEMIS (Editor), Artemis Conference 2005 and Mirror Group Meeting, Paris.
Dote, Y., 1990. Servo Motor and Motion Control Using DSP. Prentice Hall
Doyle, J.C., 1984. Advances in Multivariable Control, Honeywell Workshop, Minneapolis, MN.

Doyle, J.C., Francis, B. and Tannenbaum, A., 1990. Feedback Control Theory. Macmillan Publishing Co.

Doyle, J.C., Glover, K., Khargonekar, P.P. and Francis, B.A., 1989. State-Space Solutions To Standard H_2 and H_∞ Control Problems. IEEE Transactions on Automatic Control, 34(8): 831-847.

Dumbraï, N.V., S. Schiaua, 2000. Possibilities To increase The Resolution of Photoelectric incremental Rotary Encoders. Materials Science in Semiconductor Processing 3(Issues 5-6): 557-561.

East, T.D., 1984. Microcomputer Data Acquisition and Control. International Journal of Clinical Monitoring and Computer, 3: 225-238.

Edwards, T.M., Pont, M.J. and Crumpler, S.S., 2004. A Test Bed For Evaluating and Comparing Designs For Embedded Control Systems, Proceedings of The UK Embedded Forum, Birmingham.

Eisenberg, B.A., Johnson, G.R., Pryor, D.V. and Mccormick, S.F., 1977. A Microprocessor-Based Control System For Solar Heated/Cooled Residential Dwellings, Joint Automatic Control Conference, pp. 1157-1162.

Eldon, C.H. (Editor), 1996. Journey To The Moon: The History of The Apollo Guidance Computer. American institute of Aeronautics & Ast

Ellis, G., 2004. Control Systems Design Guide, Using Your Computer To Understand and Diagnose Feedback Controllers. Elsevier Academic Press.

Embree, P.M., 1995. C Algorithms For Real-Time DSP. Pearson Education.

Emura, T., L. Wang, 2000. A High-Resolution interpolator For incremental Encoders Based on The Quadrature PLL Method. IEEE Transactions on industrial Electronics, 47(1): 84-90.

Erickson, K.T., 1996. Programmable Logic Controllers. IEEE Potentials, 15(1): 14-17.

Eun-Chan, P., L. Hyuk, C. Chong-Ho, 2003. Position Control of X-Y Table At Velocity Reversal Using Presliding Friction Characteristics. IEEE Transactions on Control Systems Technology, 11(1): 24-31.

Faa-Jeng, L. and Po-Hung, S., 2006. Adaptive Fuzzy-Neural-Network Control For A DSP-Based Permanent Magnet Linear Synchronous Motor Servo Drive. IEEE, Transactions on Fuzzy Systems, 14(4): 481 - 495.

Faggin, F., Hoff, M.E., Mazor, S. and Shima, M., 1996. The History of The 4004. Micro, IEEE, 16(6): 10-20.

Farrar, F. and Eidens, R., 1980. Microprocessor Requirements For Implementing Modern Control Logic. IEEE Transactions on Automatic Control, 25(3): 461- 468.

Fernandes, S.M.M. and Maciel, P.R.M., 2003. Reliability Evaluation For Dependable Embedded System Specifications: An Approach Based on DSPN. In: IEEE (Editor), First ACM and IEEE international Conference on Formal Methods and Models For Co-Design, Proceedings., pp. 172-179.

Fisher, J.A., Faraboschi, P. and Young, C. (Editors), 2004. Embedded Computing A VLIW Approach To Architecture, Compilers and Tools Morgan Kaufmann.
Fobel, C., Grewal, G. and Morton, A., 2007. Using Hardware Acceleration To Reduce FPGA Placement Times. In: IEEE (Editor), Electrical and Computer Engineering, CCECE 2007, Canada, pp. 647 - 650.
Forsythe, W. and Goodall, R.M., 1991. Digital Control, Macmillan New Electronics.
Francis, B.A., 1987. A Course in H∞ Control Theory. Springer-Velag
Franklin, G., Powell, D. and Emami-Naeini, A., 1993. Feedback Control of Dynamic Systems.
Gaetano, B. and Roy, W., 2000. Embedded Computation Meets The World Wide Web. Commun. ACM, 43(5): 59-66.
Gaixin, D. Et Al., 2002. The Rotational inverted-Pendulum Based on DSP Controller. In: IEEE (Editor), 4th World Congress on intelligent Control and Automation, pp. 3101-3105
Ganssle, J. and Barr, M., 2003. Embedded Systems Dictionary. In: R. Sooby (Editor). CMP Books.
Gill, A., 1998. The Automation Evolution on The Threshold of A New Era. In: IEEE (Editor), Cement industry Technical Conference, 1998. 40th Conference Record. 1998 IEEE/PCA, pp. 403 - 410.
Godfrey, C.O., 2005. Mechatronics Principles and Applications. Elsevier.
Goksel, K., Knowles, K.A., Parrish, E.A. and Moore, J.W., 1975. An intelligent industrial Arm Using A Microprocessor. IEEE Transactions on industrial Electronics and Control instrumentation, IECI-22(3): 309-314.
Graaf, B., Lormans, M. and Toetenel, H., 2002. Software Technologies For Embedded Systems: An industry inventory Proceedings of The 4th international Conference on Product Focused Software Process Improvement Springer-Verlag, pp. 453 - 465.
Grant, D.M., P. B., Denver, 1990. Memory, Control and Communications Synthesis For Scheduled Algorithms. In: IEEE (Editor), Design Automation Conference, 1990. Proceedings. 27th ACM/IEEE, pp. 162 - 167.
Graupe, D., Magnussen, J. and Beex, A., 1978. A Microprocessor System For Multifunctional Control of Upper-Limb Prostheses Via Myoelectric Signal Identification. IEEE Transactions on Automatic Control, 23(4): 538 - 544.
Grottke, M. and Trivedi, S.K., 2007. Fighting Bugs: Remove, Retry, Replicate, and Rejuvenate. IEEE Computer, 40(2): 107-109.
Gu, D.-W., Petkov, P.H. and Konstantinov, M.M., 2005. Robust Control Design With MATLAB. Springer.
Guidi, G., H. Kubota, Y. Hori, 1997. Induction Motor Control For Electric Vehicle Application Using Lowresolution Position Sensor and Sensorless Vector Control Techique, Power Conversion Conference, pp. 937-942.

Gully, S. and Coleman, N., 1981. Microcomputer Control Algorithms For Weapon Pointing and Stabilization. IEEE Control Systems Magazine, 1(3): 22-28.

Hagiwara, N., Y. Suzuki, H. Murase, 1992. A Method of Improving The Resolution and Accuracy of Rotary Encoders Using A Code Compensation Technique. IEEE Transactions on instrumentation and Measurement 41(1): 98 - 101.

Hammarström, J. and Nilsson, J., 2006. A Comparison of Three Code Generators For Models Created in Simulink. Msc Thesis, Chalmers University of Technology, Göteborg, Sweeden.

Hanselman, D.C., R. E. Thibodeau, D. J. Smith 1989. Variable-Reluctance Resolver Design Guidelines. In: IEEE (Editor), IECON '89., 15th Annual Conference on industrial Electronics Society, pp. 203-208.

Hebert, B., M. Brule, L. A. Dessaint, 1993. A High Efficiency interface For A Biphase incremental Encoder With Error Detection [Servomotor Control]. IEEE industrial Electronics, 40(1): 155-156.

Heckenthaler, T. and Engell, S., 1994. Approximately Time-Optimal Fuzzy Control of A Two-Tank System. IEEE Control Systems

Helton, J.W., M. R. James, 1995. Reduction of Controller Complexity in Nonlinear H∞ Control. In: IEEE (Editor), Proceedings of The 34th IEEE Conference on Decision and Control, pp. 2233-2238.

Henkel, J., Hu, X.S. and Bhattacharyya, S.S., 2003. Taking on The Embedded System Design Challenge. IEEE Computer Society, 36(4): 35 - 37.

Henriksson, D., 2006. Resource−Constrained Embedded Control and Computing Systems. PhD Thesis, Lund institute of Technology, Lund.

Horowitz, I., P. Rosenbaum, 1975. Non-Linear Design For Cost of Feedback Reduction in Systems With Large Parameter Uncertainty. International Journal in Control, 21(6): 997-1001.

Huang, Y. and Yasunobu, S., 2000. A General Practical Design Method For Fuzzy PID Control From conventional PID Control. In: IEEE (Editor), The Ninth IEEE international Conference on Fuzzy Systems, San Antonio, TX, USA, pp. 969-972.

Hughes, Z., Pont, M.J. and Ong, H.L.R., 2005. Design and Evaluation of A "Time-Triggered" Microcontroller, DATE 2005 (PhD Forum), Munich, Germany

Istepanian, R.S.H. and Whidborne, J.F., 2001. Digital Controller Implementation and Fragility.

Jackson, A., 1997. A New Microcontroller With Fuzzy inference instructions Simplifies Controller Designs. In: IEEE (Editor), Aerospace Conference, pp. 491-503.

Jang, J., 1993. ANFIS: Adaptive-Network-Based Fuzzy inference System. IEEE Transactions on Systems, Man and Cybernetics, 23: 665-685.

Jinwoo, K., Bernard, P. and Zeigler, J., 1996. Hierarchical Distributed Genetic Algorithms: A Fuzzy Logic Controller Design Application. Intelligent Systems, IEEE Computer Society.

Jones, A.H., Ajlouni, N., Lin, Y.-C., Kenway, S.B. and Uzam, M., 1996. Genetic Design of Robust PID Plus Feedforward Controllers. In: IEEE (Editor), Emerging Technologies and Factory Automation, pp. 267-271.

Joshi, S.S., 1999. The Need For A Systems Perspective in Control Theory and Practice. Control Systems Magazine, IEEE, 19(6): 56-63.

Kanagaraj, N., Sivashanmugam, P. and Paramasivam, S., 2008. Fuzzy Coordinated PI Controller: Application To The Real-Time Pressure Control Process. Hindawi Journal on Advances in Fuzzy Systems(691808).

Kaul, S.K., Koul, R., Bhat, C.L., Kaul, I.K. and Tickoo, A.K., 1997. Use of A 'Look-Up' Table Improves The Accuracy of A Low Cost Resolver-Based Absolute Shaft Encoder. Measurement Science & Technology, 8(3): 329-331.

Kawaji, S. and Kanazawa, K., 1991. Control of Double inverted Pendulum With Elastic Joint. In: IEEE (Editor), intelligent Robots and Systems, pp. 946 - 951

Kelly, R. and Santibanez, V., 2003. Control De Movimiento De Robots Manipuladores.

Key, S.A., Pont, M.J. and Edwards, S., 2003 Implementing Low-Cost TTCS Systems Using Assembly Language, Proc. Eighth European Conf. Pattern Languages of Programs pp. 667-690.

Kojima, T., Kikuchi, Y., Seki, S. and Wakiwaka, H., 2004. Study on High Accuracy Optical Encoder With 30 Bits. In: IEEE (Editor), Workshop on Advanced Motion Control, pp. 493- 498.

Korn, J.E., Carroll, E., Johnson, R. and Kullack, B., 1981. Microcomputer Implementation of Control Algorithms For Weapon Pointing and Stabilization, Proceedings of The 1981 Army Numerical Analysis and Computers.

Kukkala, P., Riihimaki, J., Hannikainen, M., Hamalainen, T.D. and Kronlof, K., 2005. UML 2.0 Profile For Embedded System Design, Design, Automation and Test in Europe 2005, pp. 710-716.

Kulkarni, A.S., M. A. El-Sharkawi, 2001. Intelligent Precision Position Control of Elastic Drive Systems. IEEE Transactions on Energy Conversion, 16(1): 26-31.

Lamb, J.W. and Whelan, H.A., 1976. Microcomputer-Based Electronic Error Corrector Bar For A Numerically-Controlled Machine Tool, United States, pp. Pages: 10.

Lange, C.F.J., Chaudron, M.R.V. and Muskens, J., 2006. In Practice: UML Software Architecture and Design Description. Software, IEEE, 23(2): 40-46.

Laplante, P.A., 2005. Real-Time System Design and Analysis. In: IEEE (Editor). Willey-INTERSCIENCE.

Laub, A., Heath, M., Paige, C. and Ward, R., 1987. Computation of System Balancing Transformations and Other Applications of Simultaneous Diagonalization Algorithms. IEEE Transactions on Automatic Control 32(2): 115 - 122.
Leach, R., 1995. PID Control With 32-Bit RISC Microprocessors. In: IEEE (Editor), IEEE Technical Applications Conference and Workshops Northcon95, pp. 178-182.
Leupers, R., 2002. Compiler Design Issues For Embedded Processors. Design & Test of Computers, IEEE, 19(4): 51 - 58.
Leveson, N. and Turner, C.S., 1993. An investigation of The Therac-25 Accidents. IEEE Computer, 26(7): 18-41.
Lewis, F.L., 1992. Applied Optimal Control and Estimation. Prentice-Hall.
Lewis, F.L., 2005. Neural Networks For Feedback Control of Robots and Dynamical Systems. In: IEEE (Editor), Neural Networks, 2005. IJCNN '05. IEEE, pp. 2717.
Li, X.J. and Meijer, G.C., 1998. A Novel Low-Cost Non-Contact Resistive Potentiometric Sensor For The Measurement of Low Speeds. IEEE, Transactions on instrumentation and Measurement, 47(3): 776-781.
Liberzon, D., 2003. Output-input Stability and Feedback Stabilization of Multivariable Nonlinear Control Systems. In: IEEE (Editor), Proceedings of The 42nd IEEE Conference on Decision and Control, Maui, Hawaii USA.
Lin, C.T., 2003. Development of U.S. Air Force intercontinental Ballistic Missile Weapon Systems. Journal of Spacecraft and Rockets, 40(4).
Lin, S.-K., 1989. Simulation Study of PD Robot Cartesian Control. In: IEEE (Editor), Proceedings 1989 international Conference on Robotics and Automation, Scottsdale, AZ, USA.
Ling, K., Yue, S. and Maciejowski, J., 2006. An FPGA Implementation of Model Predictive Control. In: IEEE (Editor), in Proceedings of The Americal Control Conference.
Lipták, B.G., 2006. Instrument Engineerings' Handbook: Process Control and Optimization. ISA.
Liu, C.L. and Layland, W.J., 1973. Scheduling Algorithms For Multiprogramming in A Hard-Real-Time Environment. Association For Computing Machinery, 20(1): 40-61.
Maciejowski, J.M., 2000. MPC: A Paradigm Shift For Automatic Control. Systems Science, 26(4): 49-59.
Maclay, D., 1997. Simulation Gets into The Loop. IEE Review, 43(3): 109-112.
Machado, J.A.T. and De Carvalho, J.L.M., 1988. A New Variable Structure Controller For Robot Manipulators. In: IEEE (Editor), IEEE international Symposium on intelligent Control, pp. 441 - 446.
Machado, J.A.T. and Galhano, A.M.S.F., 1995. Benchmarking Computer Systems For Robot Control. IEEE Transactions on Education, 38(3): 205-210.

Magana, M.E. and Holzapfel, F., 1998. Fuzzy-Logic Control of An inverted Pendulum With Vision Feedback. IEEE Transactions on Education, 41(2): 165 - 170.
Maples, D.M., 1975. Floating Point Package For Intel 8008 and 8080 Microprocessors. US Energy Research and Development.
Martin, G., Lavango, L. and Louis-Guering, J., 2001. Embedded UML: A Merger of Real-Time UML and Co-Design, 9th international Symposium on Hardware/Software Codesign, Copenhagen, Denmark, pp. 23-28.
Maxwell, J.C., 1868. On Governors. Proceeding of The Royal Society, 16(100): 270–283.
Mersten, G.S., 1979. Microprocessors in Aerospace Applications. In: IEEE (Editor), Compcon pp. 264- 269.
Minorsky, N., 1922. Directional Stability and Automatically Steered Bodies. Journal of American Society of Naval Engineers, 34: 280.
Moore, G., 1979. VLSI: Some Fundamental Challenges. IEEE Spectrum 16(30).
Morse, S.P., Raveiel, B.W., Mazor, S. and Pohiman, W.B., 1980. Intel Microprocessors-8008 To 8086. IEEE Computer, 13(10): 42-60.
Moyer, B., 2001. Low-Power Design For Embedded Processors. Proceedings of The IEEE, 89(11): 1576-1587.
Mrad, F., El-Hassan, N., Mahmoud, S.E., Alawieh, B. and Adlouni, F., 2000. Real-Time Control of Free-Standing Cart-Mounted inverted Pendulum Using Labview RT. In: IEEE (Editor), industry Applications.
Mwelwa, C., 2006. Development and Assessment of A Tool To Support Pattern-Based Code Generation of Time-Triggered (TT) Embedded System. PhD Thesis, University of Leicester, Leicester.
Nair, J., Lu, N., Atraya, P. and Reuter, J., 2004. Analysis and Comparison of 3 Code Generations Tools, Proceedings of SAE international Conference, pp. 129-133.
Nakamura, M., Goto, S. and Kyura, N., 2004. Mechatronic Servo System Control. Springer.
Napper, S., 1998. Embedded-System Design Plays Catch-Up. IEEE Computer, 31(8): 120 - 118-19.
Nomura, H., Hayashi, I. and Wakami, N., 1996. A Learning Method of Fuzzy inference Rules By Descendent Method. In: IEEE (Editor), Fuzzy Systems, IEEE international Conference pp. 203-210.
Nossal, R. and Lang, R., 2002. Model-Based System Development. Micro, IEEE, 22(4): 56-63.
Ogata, K., 2002. Modern Control Engineering. Prentice Hall.
Ogawa, Y., O. Kiyoshi, 2001. Speed Observer Based Speed Control For PM Motor Having A Low Resolution Encoder. Papers of Technical Meeting on industrial instrumentation and Control, 2C-01(01-26): 79-84.
Orlosky, S., 1996. Improve Encoder Performance, PT Design Magazine, pp 43-46.
Palopoli, L., C. Pinillo, A. Sangraban, E. Laurent, A. Sangiovanni, A. Bicchi 2002. Synthesis of Robust Control Systems Under Resource Constraints,

Proceedings of Hybrid Systems: Computation and Control 2002, Stanford pp. 337-350.
Panda, S.P., A. P. Engelmann, 2003. Robust Position Control in Cassette Type Tape Drives, American Control Conference, 2003, pp. 1356-1361
Papageorgiou, G., K. Glover, 1997. A Systematic Procedure For Designing Non-Diagonal Weights To Facilitate H-infinity Loop Shaping. In: IEEE (Editor), Decision and Control, San Diego, pp. 2127-2132.
Parker, R.S., Doyle, F.J.I. and Peppas, N.A., 1999. A Model-Based Algorithm For Blood Glucose Control in Type I Diabetic Patients. IEEE Transactions on Biomedical Engineering, 46(2): 148-157.
Passino, K. and Yurkovich, S., 1998. Fuzzy Control. Addison-Wesley
Pavlica, V. and Petrovacki, D., 1998. An Application of PID-Fuzzy Hybrid Controller. Control Applications, 1998. Proceedings of The 1998 IEEE, 1(1): 629-632.
Phatrapornnant, T. and Pont, J.M., 2006. Reducing Jitter in Embedded Systems Employing A Time-Triggered Software Architecture and Dynamic Voltage Scaling. IEEE Transactions on Computer, 55(2): 113-124.
Pinnepalli, S., Jinpyo, H., Ramanujam, J. and Carver, D.L., 2007. Code Size Optimization For Embedded Processors Using Commutative Transformations. In: IEEE (Editor), Embedded and Real-Time Computing Systems and Applications, 2007. RTCSA 2007. 13th IEEE international Conference on, pp. 409 - 416.
Piotr-Jastrzebski, R., 2007. Design and Implementation of FPGA-Based LQ Control of Active Magnetic Bearings, Lappeenrania University of Technology.
Polycarpou, M., 1996. Stable Adaptive Neural Control Scheme For Nonlinear Systems. Automatic Control, IEEE Transactions.
Pont, M.J., 2001. Patterns For Time-Triggered Embedded Systems, Building Reliable Applications With 8051 Family of Microcontrollers.
Pont, M.J. and Banner, P.M., 2002. Designing Embedded Systems Using Patterns: A Case Study Journal of Systems and Software, ELSEVIER, 71(3): 201-213.
Pont, M.J. and Ong, H.L.R., 2002. Using Watchdog Timers To Improve The Reliability of TTCS Embedded Systems. In: P.A.S. Hruby, K. E. (Editor), Proceedings of The First Nordic Conference on Pattern Languages of Programs, pp. 159-200.
Postlethwaite, I.S., Tsai, M.C. and Gu, D.W., 1990. Weighting Function Selection. In: IFAC (Editor), 11th IFAC World Congress, Tallin Estonia, pp. 104-109.
Pozzi, L. and Paulin, P.G., 2007. A Future of Customizable Processors: Are We There Yet? Design, Automation & Test in Europe Conference: 1-2.
Pratt, W.C. and Brown, F.M., 1975. Automated Design of Microprocessor-Based Controllers. IEEE Transactions on industrial Electronics and Control instrumentation, , IECI-22(3): 273-279.

Prisaznuk, P.J., 2003. Software Management - ARINC 653 Application Software interface, ARINC industry Activities.
Proctor, M.F., P. W. Shackleford 2001. Real-Time Operating System Timing Jitter and Its Impact on Motor Control, Proceedings of The SPIE Sensors and Controls For intelligent Manufacturing II, pp. 10-16.
Quaranta, G. and Mantegazza, P., 2001. Using MATLAB-Simulink RTW To Build Real Time Control Applications in User Space With RTAI-LXRT. In: IEEE (Editor), Realtime Linux Workshop, Milano
Raghunath, K.J. and Parhi, K.K., 1994. Fixed and Floating Point Error Analysis of QRD-RLS and STAR-RLS Adaptive Filters. In: IEEE (Editor), Acoustics, Speech, and Signal Processing, 1994. ICASSP-94, pp. 81-84.
Rajulu, R.G. and Rajaraman, V., 1984. Estimation of Execution Times of Process Control Algorithms on Microcomputers. IEEE Transactions on industrial Electronics, IE-31(1): 56-60.
Rauta, A., Woudstra, J.B., Deleroi, W. and Van Beijnhem, E., 1996. Industrial Microcontrollers [For Electric Drives]. In: IEEE (Editor), international Conference on Power Electronics, Drives and Energy Systems For industrial Growth, pp. 991-996.
Reininger, T., F. Welker, V. M. Zeppelin 2006. Sensors in Position Control Applications For industrial Automation. Elsevier Sensors and Actuators, 129(1-2): 270-274.
Reyes, F. and Kelly, R., 2001. Experimental Evaluation of Model-Based Controllers on A Direct-Drive Robot Arm. Mechatronics, Vol. 11.
Reynaert, P., 2008. Embedded System Research in Europe. In: I.S.A.M. European Comission (Editor), Cyber-Physical Systems Workshop.
Ridgeway, R.A., 1975. Microprocessor Utilization in Hydraulic Open-Die Forge Press Control. IEEE Transactions on industrial Electronics and Control instrumentation, , IECI-22(3): 307-309.
Rovatti, R., Guerrieri, R. and Baccarani, G., 1995. An Enhanced Two-Level Boolean Synthesis Methodology For Fuzzy Rules Minimization, IEEE Transactions on Fuzzy Systems.
Ruspini, E., 1969. A New Approach To Clustering. Information and Control, 15(1): 22-32.
Sammet, J.E., 1969. Programming Languages: History and Fundamentals. Prentice-Hall.
Sangiovanni-Vincentelli, A.L. and Pinto, A., 2005. Embedded System Education: A New Paradigm For Engineering Schools? in: IEEE (Editor), Workshop on Embedded Systems Education, pp. 5-14.
Schatz, B. Et Al., 2003. CASE Tools For Embedded Systems. TUM-I0309, Technical University of Munich.
Schlett, M., 1998. Trends in Embedded-Microprocessor Design. Computing, 31(8): 44-49.
Schulz, S., Rozenblit, J.W., Mrva, M. and Buchenriede, K., 1998. Model-Based Codesign. Computer, 31(8): 60-67.

Seim, T.A., 1975. Automation of A Brazing Process With An Intel 8008 Microprocessor. IEEE Transactions on industrial Electronics and Control instrumentation, , IECI-22(3): 303-307.
Selekwa, M.F., E. G. Collins Jr., 2006. Design of Multi-objective H∞ Loop-Shaping SISO Control Systems By Fuzzy Logic. IEEE, Control and intelligent Systems, , 34(1): 45 - 56
Sevcik, R., 2006. Future FPGA Technologies, in Partnership With Universities, Proceedings of The 19th international Conference on VLSI Design Held Jointly With 5th international Conference on Embedded Systems Design. IEEE Computer Society.
Sheng, Q., Xianyi, Z., Changhong, W., Gao, X.Z. and Zilong, L., 2002. Design and Implementation of An Adaptive PID Controller Using Single Neuron Learning Algorithm. In: IEEE (Editor), Proceedings of The 4th World Congress on intelligent Control and Automation.
Short, M. and Pont, M.J., 2005. Hardware in The Loop Simulation of Embedded Automotive Control System. Intelligent Transportation Systems, 2005. Proceedings. 2005 IEEE: 426 - 431.
Shouling, H. and Xuping, X., 2007. Hardware/Software Co-Design of An ADALINE Based Adaptive Controller For A DC Motor. In: IEEE (Editor), Thirty-Ninth Southeastern Symposium on System Theory, pp. 164 - 168.
Simunic, T., Benini, L. and De Micheli, G., 2001. Energy-Efficient Design of Battery-Powered Embedded Systems. IEEE Transactions on Very Large Scale integration (VLSI) Systems, 9(1): 15-28.
Skogestad, S. and Postlethwaite, I., 2005. Multivariable Feedback Control Analysis and Design. Wiley.
Slomka, F., Dorfel, M., Munzenberger, R. and Hofmann, R., 2000. Hardware/Software Codesign and Rapid Prototyping of Embedded Systems. IEEE on Design & Test of Computers, 17(2): 28-38.
Song, I., Karray, F. and Li, H., 2005. A Real-Time Scheduler Design For Fuzzy Logic Controller. In: I.I. Conference (Editor), Mechatronics and Automation, pp. 2037 - 2042
Spong, M., Thorp, J. S., Kleinwaks, J. M., 1987. Robust Microprocessor Control of Robot Manipulators. Automatica (Journal of IFAC), 23(3): 373-379.
Spong, M. and Vidyasagar, M., 1989. Robot Dynamics and Control. John Wiley & Sons.
Standards Coordinating Committee of The IEEE Computer Society, U., 1990. IEEE Standard Glossary of Software Engineering Terminology. IEEE Computer Society.
Storey, N., 1996. Safety-Critical Computer Systems. Addison-Wesley.
Sun, Y.-L. and Meng, J., 2004. Hybrid Fuzzy Control of Robotics Systems. IEEE Transactions on Fuzzy Systems, 12(6): 755-765.
Sysml, P., 2005. Sysml Open Source Project. In: http://www.sysml.org (Editor).

Takahashi, E.T. and Wang, L., 2000. A High-Resolution interpolator For incremental Encoders Based on The Quadrature PLL Method. IEEE Transactions on industrial Electronics, 47(1): 84-90.
Tan, K.K., X. Huixing, H. L. Tong, 2002. New interpolation Method For Quadrature Encoder Signals. IEEE Transactions on instrumentation and Measurement, 51(5): 1073- 1079.
Tang, K.S., Man, F.K., Chen, G. and Kwong, S., 2001. An Optimal Fuzzy PID Controller. IEEE Transactions on industrial Electronics, 48(4): 757-765.
Teng, F.C., 2000. Real-Time Control Using MATLAB Simulink. In: IEEE (Editor), IEEE international Conference Systems, Man and Cybernetics, pp. 2697-2702.
Vanderperren, Y. and Dehaene, W., 2005. UML 2 and Sysml: An Approach To Deal With Complexity in Soc/Noc Design, Design, Automation and Test in Europe, pp. 716-718.
Venkatachalam, A. and Franz, M., 2005. Power Reduction Techniques For Microprocessor Systems ACM Computing Surveys 37(3): 195 - 237.
Veysel, G. Et Al., 2001. The RCS Handbook: Tools For Real Time Control Systems Software Development
Wang, H.-C., 1981. Sampling Period and Stability Analysis For The Microcomputer-Based Motor Control Systems. IEEE Transactions on industrial Electronics and Control instrumentation IECI-28(2): 98-102.
Wang, L.-X., 1998. Universal Approximation By Hierarchical Fuzzy Systems. Fuzzy Sets Syst. Elsevier North-Holland, 93 (2): 223-230.
Wang, R. and Yang, S., 2004. The Design of A Rapid Prototype Platform For ARM Based Embedded System. Transactions on Consumer Electronics, 50(2).
Whidborne, J.F. and Yang, J.-B., 2002. Multi-objective Design of Low Complexity Digital Controllers. In: IEEE (Editor), international Symposium on Computer Aid Control Systems Design pp. 27-32.
Williamson, D., 1991. Digital Control and Implementation Finite Wordlength Considerations, Prentice Hall.
Wolf, W., Ozer, B. and Lv, T., 2002. Smart Cameras As Embedded Systems. IEEE Computer, 35(9): 48- 53.
Wootaik, L., Minsuk, S. and Myoungho, S., 2004. Target-Identical Rapid Control Prototyping Platform For Model-Based Engine Control Proceedings of The I MECH E Part D Journal of Automobile Engineering(D7): 755-765.
Wörnle, F. and Murillo-Garcia, R., 2002. A MATLAB Toolbox For Real-Time Control Using C167 Microcontrollers. In: IEEE (Editor), Control and Applications, Cancun Mexico.
WSTS, 2005. Embedded Systems Statistics. In: http://www.wsts.org/ (Editor). World Semiconductor Trade Statistics, .
Yamakawa, T., 1992. A Fuzzy Programmable Logic Array (Fuzzy PLA). In: IEEE (Editor), international Conference on Fuzzy Systems, pp. 459-465.

Youbin, P., Vrancic, D. and Hanus, R., 1996. Anti-Windup, Bumpless, and Conditioned Transfer Techniques For PID Controllers. IEEE Control Systems Magazine, 16(4): 48-57.
Zadeh, L.A., 1965. Fuzzy Sets. Information and Control, 8.
Zames, G., 1976. Feedback and Complexity, Special Plenary Lecture Addendum. In: IEEE (Editor), Decision and Control.
Zames, G., 1981. Feedback and Optimal Sensitivity: Model Reference Transformations, Multiplicative Seminorms, and Approximate inverses. IEEE Transactions on Automatic Control, 26(2): 301-320.
Zhou, K., J. C. Doyle, K. Glover, 1996. Robust and Optimal Control. Prentice-Hall.
Zhou, S.-L., Han, P., Dong-Feng, W., Wang, W.-Y. and Zhang, L.J., 2003. Fuzzy-Neural Net Based Control Strategy For Robot Manipulator Trajectory Tracking. In: IEEE (Editor), Machine Learning and Cybernetics, pp. 596-601.

APPENDIX A

Programs for the inverted pendulum testbed using LPC2129 ARM microcontroller

main.c

```
/*****************************************************************************************************/
/*  Using Keil uVision/ARM and the Keil CA ARM Compiler board MC2100                                 */
/*  EMBEDDED SYSTEMS LABORATORY                                                                      */
/*  UNIVERSITY OF  LEICESTER                                                                         */
/*  ENGINEERING DEPARTMENT                                                                           */
/*****************************************************************************************************/
/* main function                                                                                     */
/* Control for the pendulum using encoder 1000 ppr                                                   */
/* SIMO control                                                                                      */
/* SERVOMOTOR PITMAM MODEL GM9234C2112-R3                                                            */
/* Author: Ricardo Bautista Quintero                                                                 */
/*                                                                                                   */
/*****************************************************************************************************/

#include <LPC21xx.H>            // Head file LPC21xx
#include <stdio.h>              // Prototype for I/O functions

#define offset_rod 218          // Offset rod position

void init_timer(void);          // Initialization of the timer
void task1(void)__irq;          // Acquisition and control task

void init_eint1(void);          // External interruption for the cart encoder Phase A
void ExtIServ1(void) __irq;
void init_eint0 (void);         // Interruption for the rod encoder Phase A and Phase B
void ExtIServ0(void) __irq;
void init_eint3 (void);         // Interruption for the rod encoder Phase A and Phase B
void ExtIServ3(void) __irq;
void init_PWM (void);           // Initialization of PWM signal (signal for the motor actuation)

void main (void)
{
        char end_flag;
        sp=0;
        end_flag=1;
        xp_max= 0;
        j=0;                // Init rod position in zero
        init_eint0();       // Initialization of external interruption 0
        init_eint1();       // Initialization of external interruption 1
        init_eint3();       // Initialization of external interruption 3
        do{
                }while(j<offset_rod);
                i=0;
                j=0;
init_timer();               // Initialization  TIMER 0
init_PWM();                 // Initialization PWM2 Module

        do{                 // Super-loop for the  real-time control process
         }while(end_flag);

}
```

External_int1.c

```
/*****************************************************************************************************/
/*  Using Keil uVision/ARM and the Keil CA ARM Compiler board MC2100                                 */
/*  EMBEDDED SYSTEMS LABORATORY                                                                      */
/*  UNIVERSITY OF  LEICESTER                                                                         */
/*  ENGINEERING DEPARTMENT                                                                           */
/*****************************************************************************************************/
/* External interruption 1                                                                           */
/* Control for the pendulum using encoder 1000 ppr                                                   */
/* SIMO control                                                                                      */
/* SERVOMOTOR PITMAM MODEL GM9234C2112-R3                                                            */
/* Author: Ricardo Bautista Quintero                                                                 */
/*                                                                                                   */
/*****************************************************************************************************/

extern float i;                  // Counter for car position
extern float j;                  // Counter for rod position

// External Interrupt 1 Service
void ExtIServ1(void) __irq
{
  if ((IOPIN0 & 0x00000400) != 0)          // Check if Phase B is ON
          {
            i=i+1;                 // Cart is moving forwards (right) increase the counter
           }else i=i-1;            // Cart is moving backwards (left) decrease the counter

  EXTINT      = 2;                 // Clear interrupt flag EINT1
  VICVectAddr = 0;                 // Acknowledge Interrupt
}

// Init External Interrupt Pin

void init_eint1 (void)
{
EXTMODE= 0x02;                                  // Edge sensitive mode on EINT1
EXTPOLAR = 0x02;                                // Raising edge sensitive
PINSEL0= PINSEL0 | 0x000000C0;                  // Enable EINT1 on GPIO_0.3
VICVectAddr1= (unsigned long) ExtIServ1;        // Set interrupt vector in VIC 0
VICVectCntl1 = 0x2F;                            // Use VIC 1 for EINT1 Interrupt
VICIntEnable = VICIntEnable | 0x00008000;       // Enable EINT1 Interrupt
}
```

Ext_int0.c

```
/*****************************************************************************************************/
/*  Using Keil uVision/ARM and the Keil CA ARM Compiler board MC2100                                  */
/*  EMBEDDED SYSTEMS LABORATORY                                                                       */
/*  UNIVERSITY OF  LEICESTER                                                                          */
/*  ENGINEERING DEPARTMENT                                                                            */
/*****************************************************************************************************/
/* External interruption 0     (implementation of quad technique for encoder readings)               */
/* Control for the pendulum using encoder 1000 ppr                                                   */
/* SIMO control                                                                                      */
/* SERVOMOTOR PITMAM MODEL GM9234C2112-R3                                                            */
/* Author: Ricardo Bautista Quintero                                                                 */
/*                                                                                                   */
/*****************************************************************************************************/

char t1=1,t2=1;                    // Toggle variables to detect falling and rising edges

// External Interrupt 0 Service
void ExtIServ0(void) __irq
{
if(t1==1)          //Toggle variable
          {t1=0;
           if ((IOPIN0 & 0x00000001) != 0)  // Read phase A (pin P0.0)
          {
            j++;              // Rod rotating in CW increase counter
           }else j--;         // Rod rotating in CCW decrease counter

          EXTPOLAR=EXTPOLAR & 0xE; // Change pulses edge detection
           }else
                    {
                    t1=1;
                    if ((IOPIN0 & 0x00000001) != 0)  // Read phase A (pin P0.0)
                              {
                               j--;              // Rod rotating in CW decrease counter
                              }else j++;         // Rod rotating in CCW decrease counter

                              EXTPOLAR=EXTPOLAR | 0X1; // Change pulses edge detection
          }

EXTINT      = 1;           // Clear interrupt flag EINT0
VICVectAddr = 0;           // Acknowledge Interrupt
}

// Init External Interrupt Pin

void init_eint0 (void) {

EXTMODE= 0x01;                                  // Edge sensitive mode on EINT0
EXTPOLAR= 0x01;                                 // Raising edge sensitive
PINSEL0 = PINSEL0 | 0x0000000C;                 // Enable EINT0 on GPIO_0.1
VICVectAddr2 = (unsigned long) ExtIServ0;       // Set interrupt vector in VIC 0
VICVectCntl2 = 0x2E;                            // Use VIC 0 for EINT0 Interrupt
VICIntEnable = VICIntEnable | 0x00004000;       // Enable EINT0 Interrupt
}
```

Ext_int3.c

```
/*****************************************************************************************************/
/*  Using Keil uVision/ARM and the Keil CA ARM Compiler board MC2100                                 */
/*  EMBEDDED SYSTEMS LABORATORY                                                                      */
/*  UNIVERSITY OF  LEICESTER                                                                         */
/*  ENGINEERING DEPARTMENT                                                                           */
/*****************************************************************************************************/
/* External interruption 3     (implementation of quad technique for encoder readings)              */
/* Control for the pendulum using encoder 1000 ppr                                                  */
/* SIMO control                                                                                     */
/* SERVOMOTOR PITMAM MODEL GM9234C2112-R3                                                           */
/* Author: Ricardo Bautista Quintero                                                                */
/*                                                                                                  */
/*****************************************************************************************************/

char t1=1,t2=1;                     // Toggle variables to detect falling and rising edges

// External Interrupt 3 Service
void ExtIServ3(void) __irq
{
 if(t2==0)          //Toggle variable
          { t2=1;

          if ((IOPIN0 & 0x00000010) != 0) // Read phase A (P0.4)
                    {
                     j--;               // Rod rotating in CCW decrease counter
                     }else j++;         // Rod rotating in CW increase counter

                    EXTPOLAR=EXTPOLAR & 0X7;          // Change pulses edge detection
                    }else
                    {
                    t2=0;
                     if ((IOPIN0 & 0x00000010) != 0)// pin P0.4
                              {
                              j++;              // Rod rotating in CW increase counter
                              }else j--;        // Rod rotating in CCW decrease counter

                              EXTPOLAR=EXTPOLAR | 0x08;// Change pulses edge detection
          }
  EXTINT      = 8;                  // Clear interrupt flag EINT3
  VICVectAddr = 0;                   // Acknowledge Interrupt
}

// Init External Interrupt Pin

void init_eint3 (void) {

EXTMODE = 0x02;                                 // Edge sensitive mode on EINT2
EXTPOLAR= 0x03;                                 // Raising edge sensitive
PINSEL0 = PINSEL0 | 0x000C0000;                 // Enable EINT3 on GPIO_0.9
VICVectAddr3 = (unsigned long) ExtIServ3;       // Set interrupt vector in VIC 0
VICVectCntl3 = 0x21 | 17;                       // Use VIC 2 for EINT3 Interrupt
VICIntEnable = VICIntEnable | 0x00020000;       // Enable EINT3 Interrupt
}
```

PWM_init.c

```
/*****************************************************************************************************/
/*  Using Keil uVision/ARM and the Keil CA ARM Compiler board MC2100                                 */
/*  EMBEDDED SYSTEMS LABORATORY                                                                      */
/*  UNIVERSITY OF  LEICESTER                                                                         */
/*  ENGINEERING DEPARTMENT                                                                           */
/*****************************************************************************************************/
/* PWM module     Signal to actuate Servomotor                                                      */
/* Control for the pendulum using encoder 1000 ppr                                                  */
/* SIMO control                                                                                     */
/* SERVOMOTOR PITMAM MODEL GM9234C2112-R3                                                           */
/* Author: Ricardo Bautista Quintero                                                                */
/*                                                                                                  */
/*****************************************************************************************************/

void PWM_Init(void)
 {
   PIN_Set_Mode(PWM_OUTPUT_pin, 1, 0);  // Set P0.07 as PWM2.

   PWMPR  = 0x00000000;         // 1:1 ratio of the PWM prescaler.
   PWMPCR = 0x00000404;         // Control mode for PWM2 output. Enables PWM2 output.
   PWMMCR = 0x00000002;         // PWMTC is reset.
   PWMMR0 = 0x400;     // Set PWMMCR0 to 1042, making the PWM period (1/PCLK) x 1024.
   PWMMR2 = 0x00;       // Clear the PWMMR2 register.
   PWMLER = 0xF;          // Enable shadow latch for PWM Match 0 - 3.
   PWMEMR = 0x00210A8E;    // Match 1 and Match 2 outputs set high.
   PWMTCR = 0x00000002;    // Reset PWMTC and PWMPR.
}
```

PID_real_time_rutine.c

```
/*****************************************************************************************************/
/*  Using Keil uVision/ARM and the Keil CA ARM Compiler board MC2100                                 */
/*  EMBEDDED SYSTEMS LABORATORY                                                                      */
/*  UNIVERSITY OF  LEICESTER                                                                         */
/*  ENGINEERING DEPARTMENT                                                                           */
/*****************************************************************************************************/
/* PID function    PD control for the rod and PID for the cart                                       */
/* Control for the pendulum using encoder 1000 ppr                                                   */
/* SERVOMOTOR PITMAM MODEL GM9234C2112-R3                                                            */
/* Author: Ricardo Bautista Quintero                                                                 */
/*****************************************************************************************************/

#include<LPC21XX.H>

#define zero 20
#define Ts 0.01                    // Sample time 10 ms
#define pulses2mts 6.8327e-5       // Constant to convert pulses to metres (cart sensor)
#define pulses2rad 1.57e-3         // Constant to convert pulses to radians (rod sensor)
#define volts2PWM 1024/19          // Constant to convert volts to bits (actuator signal)

void init_timer(void);

float i;   // Counter for  pulses (cart encoder)
float j;   // Counter for  pulses (rod encoder)

int time=0;        // Time variable
char end_flag;   // Variable to cancel the process

/*************************************/
/* variables for sub-task1           */
/*************************************/
int sub_task1=0;
int time_task1=50;
float pcart_mts=0;
float teta_rad=0;

/*************************************/
/* variables for sub-task2           */
/*************************************/
int sub_task2=0;
int time_task2=100;
float vcart=0;
float vrod=0;
float vcartf=0;
float vrodf=0;
float pcart_mts_old=0;
float teta_rad_old=0;
float sp=0;
float error=0;
float kpc=-4.1813;         // position of the cart
float kpr=31.38;           // position of the rod
float kdc=-15.28;          // velocity of the cart
float kdr=7.5743;          // velocity of the rod
float kic=0.5;             // integral control
float Iz=0;                // Integral accumulator
float Izold=0;             // Integral history
float alfa=0.85;           // Digital filter coefficient
```

```
/*************************************/
/* variables for sub task3           */
/*************************************/
int sub_task3=0;
int time_task3=200;

void task1(void)__irq   // Real-time task1
{
float u=0;
sub_task1++;
        if(sub_task1>=time_task1)
                                        {
                                        // This segment is executed every 5 ms
                                         pcart_mts=i*pulses2mts;
                                         teta_rad=j*pulses2rad;
                                         sub_task1=0;
                                         }
sub_task2++;
        if(sub_task2>=time_task2)
                                        {
                                        // This segment is executed every 10 ms
                                        vcart=(pcart_mts-pcart_mts_old)/Ts;
                                        vcartf=alfa*vcartf+(1-alfa)*vcart;
                                        vrod=(teta_rad-teta_rad_old)/Ts;
                                        vrodf=alfa*vrodf+(1-alfa)*vrod;
                                        pcart_mts_old=pcart_mts;
                                        teta_rad_old=teta_rad;
                                        Iz=Izold+kic*Ts*(sp-pcart_mts);
                                        u=-kpr*teta_rad-kdr*vrodf+kpc*(sp-pcart_mts)-kdc*vcartf-Iz;
                                        u=u*volts2PWM;
                                        Izold=Iz;
                                        if(u <  0) IOSET1=0X010000;else  IOCLR1=0X010000;
                                        if(u<0)u=-u;
                                        if(u>1020)u=1020;       // Saturacion
                                        PWMMR2 = u+zero;        // Salida de control
                                        PWMLER = 0x4;           // Enable Shadow latch
                                        time++;
                                        sub_task2=0;
                                        }
sub_task3++;
        if(sub_task3>=time_task3)
                                        {
                                        // This segment is executed every 20 ms
                                        if(pcart_mts>0.3){end_flag=0;return;}
                                        if(pcart_mts<-0.3){end_flag=0;return;}
                                        if(time>=1200){end_flag=0;return;}
                                        sub_task3=0;
                                         }
T0IR=1; VICVectAddr=0;
}

void init_timer(void)
{
T0MR0=2999;             //Number of cycles to occur an interruption 3000-1=1 int every 0.1ms
T0MCR=3;                //Init interruption
T0TCR=1;                //Enable timer 0
VICVectAddr0=(unsigned long)task1;
VICVectCntl0=0x20 | 4;
VICIntEnable = VICIntEnable | 0x00000010;       // Enable interruption of timer 0
}
```

LQR_real_time_rutine.c

```
/*****************************************************************************************************/
/*  Using Keil uVision/ARM and the Keil CA ARM Compiler board MC2100                                  */
/*  EMBEDDED SYSTEMS LABORATORY                                                                       */
/*  UNIVERSITY OF  LEICESTER                                                                          */
/*  ENGINEERING DEPARTMENT                                                                            */
/*****************************************************************************************************/
/* LQR function feedback: k1 cart position, k2 rod position, k3 cart velocity, k4 rod velocity */
/* Control for the pendulum using encoder 1000 ppr                                                   */
/* SERVOMOTOR PITMAM MODEL GM9234C2112-R3                                                            */
/* Author: Ricardo Bautista Quintero                                                                 */
/*****************************************************************************************************/
#include<LPC21XX.H>

#define zero 20
#define Ts 0.01                    // Sample time 10 ms
#define pulses2mts 6.8327e-5       // Constant to convert pulses to metres (cart sensor)
#define pulses2rad 1.57e-3         // Constant to convert pulses to radians (rod sensor)
#define volts2PWM 1024/19          // Constant to convert volts to bits (actuator signal)

/*************************************/
/* variables para sub task1          */
/*************************************/
int sub_task1=0;
int time_task1=50;
float pcart_mts=0;
float teta_rad=0;
/*************************************/
/* variables para sub task2          */
/*************************************/
int sub_task2=0;
int time_task2=100;
float vcart=0;
float vrod=0;
float vcartf=0;
float vrodf=0;
float pcart_mts_old=0;
float teta_rad_old=0;
float sp=0;
float error=0;

/*************************************/
/*   LQR Gains                       */
/*************************************/
float k1=-4.18;              //position of the cart
float k2=-33.17;             //position of the rod
float k3=-18.09;             //velocity of the cart
float k4=-7.89;              //velocity of the rod

float k0=0.1485;             //feedback of the input
float w0=0;                  // output of control
float alfa=0.95;

/*************************************/
/* variables for sub task3           */
/*************************************/

int sub_task3=0;
int time_task3=200;
```

```
void task1(void)__irq   // Real-time task1
{
float u=0;
sub_task1++;
        if(sub_task1>=time_task1)
                                        {
                                        // This segment is executed every 5 ms
                                         pcart_mts=i*pulses2mts;
                                         teta_rad=j*pulses2rad;
                                         sub_task1=0;
                                         }
sub_task2++;
        if(sub_task2>=time_task2)
                                        {
                                        // This segment is executed every 10 ms
                                        vcart=(pcart_mts-pcart_mts_old)/Ts;
                                        vcartf=alfa*vcartf+(1-alfa)*vcart;
                                        vrod=(teta_rad-teta_rad_old)/Ts;
                                        vrodf=alfa*vrodf+(1-alfa)*vrod;
                                        pcart_mts_old=pcart_mts;
                                        teta_rad_old=teta_rad;
                                        u=k1*(sp-pcart_mts)-k2*teta_rad-k3*vcartf-k4*vrodf+k0*w0;
                                        u=u*volts2PWM;
                                        Izold=Iz;
                                        if(u <  0) IOSET1=0X010000;else  IOCLR1=0X010000;
                                        if(u<0)u=-u;
                                        if(u>1020)u=1020;       // Saturacion
                                        PWMMR2 = u+zero;        // Salida de control
                                        PWMLER = 0x4;           // Enable Shadow latch
                                        time++;
                                        sub_task2=0;
                                        }
sub_task3++;
        if(sub_task3>=time_task3)
                                        {
                                        // This segment is executed every 20 ms
                                        if(pcart_mts>0.3){end_flag=0;return;}
                                        if(pcart_mts<-0.3){end_flag=0;return;}
                                        if(time>=1200){end_flag=0;return;}
                                        sub_task3=0;
                                         }
T0IR=1; VICVectAddr=0;
}
```

FLC_real_time_rutine.c

```
/*****************************************************************************************************/
/*  Using Keil uVision/ARM and the Keil CA ARM Compiler board MC2100                                 */
/*  EMBEDDED SYSTEMS LABORATORY                                                                      */
/*  UNIVERSITY OF  LEICESTER                                                                         */
/*  ENGINEERING DEPARTMENT                                                                           */
/*****************************************************************************************************/
/* FLC_function                                                                                      */
/* Control for the pendulum using encoder 1000 ppr                                                   */
/* SERVOMOTOR PITMAM MODEL GM9234C2112-R3                                                            */
/* Author: Ricardo Bautista Quintero                                                                 */
/*****************************************************************************************************/
#include<LPC21XX.H>

#define zero 20
#define Ts 0.01                 // Sample time 10 ms
#define pulses2mts 6.8327e-5    // Constant to convert pulses to metres (cart sensor)
#define pulses2rad 1.57e-3      // Constant to convert pulses to radians (rod sensor)
#define volts2PWM 1024/19       // Constant to convert volts to bits (actuator signal)

#define num_max_MSF     7       // Maximum number of membership functions
#define num_max_inputs  4       // Maximum number of inputs
#define num_max_rules   50      // Maximum number of rules
#define Coord_Points    4       // Coordinates of MSF

#define num_inputs      4       // Inputs
#define num_outputs     1       // Outputs
#define num_rules1      18      // Rules for rules for set 1
#define num_sets        1       // Number of sets
#define num_conse1      5       // Number of consequents
#define num_ante1       5       // Number of antecedents 1
#define num_ante2       3       // Number of antecedents 2
#define num_ante3       5       // Number of antecedents 3
#define num_ante4       3       // Number of antecedents 4


int     num_input_mfs[4] = { num_ante1,num_ante2,num_ante3,num_ante4 };
int     num_rules[1]={num_rules1};
int     num_conse[1]={num_conse1};

float crisp_in[num_max_inputs];
float fuzzy_in[num_max_inputs][num_max_MSF];
char Membership[num_max_MSF];

void fuzzy(float data_in,int num_var);
float rules_eval(char set_num);

float   inmem_points[num_inputs][num_max_MSF][Coord_Points] =
{

 {  { -30.000000, -30.000000, -10.000000, -5.000000 },
    { -10.000000, -5.000000, -5.000000, 0.000000 },
    { -5.000000, 0.000000, 0.000000, 5.000000 },
    { 0.000000, 5.000000, 5.000000, 10.000000 },
    { 5.000000, 10.000000, 30.000000, 30.000000 }

  },
```

```
{
   {-5.000000, -5.000000, -3.000000,0.000000 },
   { -3.000000, 0.000000, 0.000000, 3.000000 },
   { 0.000000, 3.000000, 5.0000000, 5.0000000 }
},
 {
   { -40.000000, -40.000000, -32.000000, -21.000000 },
   { -32.000000, -21.000000, -21.000000, 0.000000 },
   { -21.000000, 0.000000, 0.000000, 21.000000 },
   { 0.000000, 21.000000, 21.000000, 32.000000 },
   { 21.000000, 32.000000, 40.000000, 40.000000 }
 },

{
   {-40.000000, -40.000000, -20.000000,0.000000 },
   { -20.000000, 0.000000, 0.000000, 20.000000 },
   { 0.000000, 20.000000, 40.000000, 40.0000000 }
}

};

float      rules[num_sets][num_rules1][5]=
{
  {
   { 2, 1, 2, 1, 2 },// ang zero & vel_ang zero then volta zero
   { 1, 1, 2, 1, 3 },// ang -    & vel_ang zero then volta +
   { 0, 1, 2, 1, 4 },// ang --   & vel_ang zero then volta ++
   { 3, 1, 2, 1, 1 },// ang +    & vel_ang zero then volta -
   { 4, 1, 2, 1, 0 },// ang ++   & vel_ang zero then volta --
   { 1, 0, 2, 1, 0 },// ang -    & vel_ang --   then volta --
   { 3, 2, 2, 1, 4 },// ang +    & vel_ang ++   then volta ++
   { 2, 0, 2, 1, 3 },// ang zero & vel_ang - then volta +
   { 2, 2, 2, 1, 1 },// ang zero & vel_ang + then volta -

   { 2, 1, 2, 1, 2 },// car zero & vel_car zero then volta zero
   { 2, 1, 1, 1, 3 },// car -    & vel_car zero then volta +
   { 2, 1, 0, 1, 3 },// car --   & vel_car zero then volta +
   { 2, 1, 3, 1, 1 },// car +    & vel_car zero then volta -
   { 2, 1, 4, 1, 1 },// car ++   & vel_car zero then volta -

   { 2, 1, 1, 0, 0 },// car -    & vel_car - then volta --
   { 2, 1, 3, 2, 4 },// car +    & vel_car + then volta ++
   { 2, 1, 2, 0, 1 },// car zero & vel_car - then volta -
   { 2, 1, 2, 2, 3 } // car zero & vel_car + then volta +
  }
};

int        num_output_mfs[1] = {num_conse1};

float      outmem_points[1][num_conse1] =
{
          {
                    { -16.000000 },
                    { -12.000000 },
                    { 0.000000 },
                    { 12.000000 },
                    { 16.000000 }
          }
};
```

```
/*************************************/
/* variables para sub task1          */
/*************************************/
int sub_task1=0;
int time_task1=50;
float pcart_cm=0;
float teta_grad=0;

/*************************************/
/* variables para sub task2          */
/*************************************/
int sub_task2=0;
int time_task2=100;
float vcart=0;
float vrod=0;
float vcartf=0;
float vrodf=0;
float pcart_cm_old=0;
float teta_grad_old=0;
float sp=0;
float error=0;

/*************************************/
/* variables for sub task3           */
/*************************************/

int sub_task3=0;
int time_task3=200;
void task1(void)__irq   // Real-time task1
{
float u=0;
sub_task1++;
        if(sub_task1>=time_task1)
                                        {
                                        // This segment is executed every 5 ms
                                        pcart_cm=i_pulses*pulses2cm;
                                        teta_grad=j_pulses*pulses2grad;
                                         sub_task1=0;
                                         }
sub_task2++;
        if(sub_task2>=time_task2)
                                        {
                                        // This segment is executed every 10 ms
                                        vcart=(pcart_cm-pcart_cm_old)/Ts;
                                        vcartf=alfa*vcartf+(1-alfa)*vcart;
                                        vrod=(teta_grad-teta_grad_old)/Ts;
                                        vrodf=alfa*vrodf+(1-alfa)*vrod;
                                        pcart_cm_old=pcart_cm;
                                        teta_grad_old=teta_grad;

                                        fuzzy(teta_grad*0.8,0);
                                        fuzzy(vrodf/22,1);
                                        fuzzy((sp-pcart_cm)/5,2);
                                        fuzzy(vcartf/2,3);
                                        u=rules_eval(0);
                                        u=u*volts2PWM;
                                        Izold=Iz;
                                        if(u <  0) IOSET1=0X010000;else  IOCLR1=0X010000;
                                        if(u<0)u=-u;
```

```
                                    if(u>1020)u=1020;       // Saturation
                                    PWMMR2 = u+zero;        // control output
                                    PWMLER = 0x4;           // Enable Shadow latch
                                    time++;
                                    sub_task2=0;
                                    }
sub_task3++;
        if(sub_task3>=time_task3)
                                    {
                                    // This segment is executed every 20 ms
                                    if(pcart_cm>35){end_flag=0;return;}
                                    if(pcart_cm<-35){end_flag=0;return;}
                                    sub_task3=0;
                                     }
T0IR=1; VICVectAddr=0;
}
```

H_inf_real_time_rutine.c

```
/*****************************************************************************************************/
/*  Using Keil uVision/ARM and the Keil CA ARM Compiler board MC2100                                 */
/*  EMBEDDED SYSTEMS LABORATORY                                                                      */
/*  UNIVERSITY OF  LEICESTER                                                                         */
/*  ENGINEERING DEPARTMENT                                                                           */
/*****************************************************************************************************/
/* H-infinity function                                                                               */
/* Control for the pendulum using encoder 1000 ppr                                                   */
/* SERVOMOTOR PITMAM MODEL GM9234C2112-R3                                                            */
/* Author: Ricardo Bautista Quintero                                                                 */
/*****************************************************************************************************/
#include<LPC21XX.H>

#define zero 20
#define Ts 0.01                    // Sample time 10 ms
#define pulses2mts 6.8327e-5       // Constant to convert pulses to metres (cart sensor)
#define pulses2rad 1.57e-3         // Constant to convert pulses to radians (rod sensor)
#define volts2PWM 1024/19          // Constant to convert volts to bits (actuator signal)

void init_timer(void);
void mul_mat(float *p1,float *p2,int row1,int col1,int row2, int col2);

float i_pulses;     // Counter for  pulses (cart encoder)
float j_pulses;     // Counter for  pulses (rod encoder)

/**************************************/
/* variables para sub task1           */
/**************************************/
int sub_task1=0;
int time_task1=50;
float pcart_mts=0;
float teta_rad=0;
/**************************************/
/* variables para sub task2           */
/**************************************/
float sp=0;

#define no_max 12          // Maximum number for matrix multiplication

float AplusB[no_max][no_max]; //matrix dummy

/*********************************************************/
/* Variables of the H-inf  control and system            */
/*********************************************************/

#define Hinf_edos       8
#define Hinf_inp        2
#define Hinf_out        1
#define Syst_edos       4
#define Syst_inp        1
#define Syst_out        2
```

```
float Akd[Hinf_edos][Hinf_edos]=

{
{2.1467374967888642000e-002,  -5.4382512659417621000e+000,  -
4.7316556248022140000e-002,  -2.0680245649531773000e-001,  -
4.7877412279279321000e+000,  -1.0534585005118013000e+000,
1.3068615786072331000e+000,  -1.1909687024866757000e+000,  },

{0.0000000000000000000e+000,  8.2321677856902509000e-001,  -
1.3667601178941859000e+000,  7.9354096162959464000e-001,  -
2.8286895076154794000e-001,  -1.7760638789758667000e+000,
4.7091573223426691000e-002,  -4.2992685039849036000e-002,  },

{0.0000000000000000000e+000,  0.0000000000000000000e+000,
1.9520731864853392000e-007,  1.4315125648472195000e-006,  -
5.4273201013730406000e-002,  -1.0816750352048157000e+000,
1.7226714168862647000e-002,  -1.5726242273056398000e-002,  },

{0.0000000000000000000e+000,  0.0000000000000000000e+000,  -
4.0398299262517641000e-008,  1.9520731864853392000e-007,
1.7832177208772043000e-001,  4.4954676426353557000e-001,  -
1.1556037096040511000e-002,  1.0550514957436356000e-002,  },

{0.0000000000000000000e+000,  0.0000000000000000000e+000,
0.0000000000000000000e+000,  0.0000000000000000000e+000,
1.0300951875194466000e+000,  4.0215619125448306000e-002,  -
6.6845295330444487000e-003,  6.1048152621330726000e-003,  },

{0.0000000000000000000e+000,  0.0000000000000000000e+000,
0.0000000000000000000e+000,  0.0000000000000000000e+000,
0.0000000000000000000e+000,  9.9582866721528462000e-001,  -
1.3291037892879550000e-003,  1.2138112584635591000e-003,  },

{0.0000000000000000000e+000,  0.0000000000000000000e+000,
0.0000000000000000000e+000,  0.0000000000000000000e+000,
0.0000000000000000000e+000,  0.0000000000000000000e+000,
9.9994999867302103000e-001,  2.4031960068303890000e-009,  },

{0.0000000000000000000e+000,  0.0000000000000000000e+000,
0.0000000000000000000e+000,  0.0000000000000000000e+000,
0.0000000000000000000e+000,  0.0000000000000000000e+000,
0.0000000000000000000e+000,  9.9900050035718735000e-001,  },
};

float Bkd[Hinf_edos][Hinf_inp]=
{

{-8.0351233645522679000e-001,  -6.6575102915912794000e+002,  },
{-4.7435850707732666000e-002,  1.2001633778504456000e+001,  },
{4.4627947172326682000e-003,  -7.7543589944093441000e+000,  },
{-3.9171858512750818000e-003,  5.1910427661786471000e+000,  },
{-2.5958906627141953000e-002,  1.4760754243892620000e+001,  },
{-1.1386376717878524000e-002,  6.2338018738758194000e+000,  },
{-5.8310723527633213000e-001,  1.1310545098890259000e-005,  },
{-1.6034939336681212000e-008,  -4.6626425200095207000e+000,  },

};
```

```
float Ck[1][Hinf_edos]={
{-5.351158135264455500000e-002,  -5.392557169641625200000e-001,  -
8.730479261360730400000e+001,  7.687422906918731100000e+002,  -
1.381442105548318300000e+002,  -4.364355446297364000000e+002,
9.001975399250554200000e+000,  -8.225732525077569700000e+000,  },
};

float Uk[Hinf_inp][1]= {
                        {0},
                        {0},
                        };

float Xk[Hinf_edos][1]={
                        {0},
                        {0},
                        {0},
                        {0},
                        {0},
                        {0},
                        {0},
                        {0},
                        };

float U[1][1];

float x[Syst_edos][1]={
                        {0}, // Cart position
                        {0}, // Rod position
                        {0}, // Cart velocity
                        {0}, // Rod velocity
                        };

/*************************************/
/* Variables for sub task3           */
/*************************************/

int sub_task3=0;
int time_task3=200;

void task1(void)__irq
{
float u=0;
int edo;
float *pa,*pb;                          // Pointers for matrix A and B

sub_task1++;                            // Execute this code every 5 ms
if(sub_task1>=time_task1){
                          pcart_mts=i_pulses*pulses2mts;
                          teta_rad=j_pulses*pulses2rad;
                         sub_task1=0;
                         }

sub_task2++;                            // Execute this code every 10ms
if(sub_task2>=time_task2){
```

```
        // ********************************************
        // Implementation of H-inf controller
        // xk=Akd*xk+Bkd*Uk
        // Uk=SP-Y;
        // Yk=Ckd*xk;
        // ********************************************
                pa=&Akd[0][0];
                pb=&Xk[0][0];

                mul_mat(pa,pb,Hinf_edos,Hinf_edos,Hinf_edos,1); //calculo de Akd*xk
                for(edo=0;edo<Hinf_edos;edo++)
                Xk[edo][0]=AplusB[edo][0];          //xk'=Ak*xk
                pa=&Bkd[0][0];
                pb=&Uk[0][0];
                mul_mat(pa,pb,Hinf_edos,Hinf_inp,Hinf_inp,1); // Obtaining Bkd*Uk
                for(edo=0;edo<Hinf_edos;edo++)
                Xk[edo][0]=Xk[edo][0]+AplusB[edo][0]; //xk'=Ak*xk+Bk*Uk

                Uk[0][0]=sp-pcart_mts; // Setpoint – cart position
                Uk[1][0]=teta_rad;  // Teta_radians

                pa=&Ck[0][0];
                pb=&Xk[0][0];
                mul_mat(pa,pb,Hinf_out,Hinf_edos,Hinf_edos,Hinf_out);
                U[0][0]=AplusB[0][0]; // Obtaining Yk Input to the system (control signal u)

                u=U[0][0];
                if(u <  0) IOSET1=0X010000;
                if(u >= 0) IOCLR1=0X010000;

                 u=u*volts2PWM;          // PWM signal
                if(u<0)u=-u;
                if(u>1020)u=1020;        // Saturation

                PWMMR2 = u+zero;      // Control output
                PWMLER = 0x4;          // Enable Shadow latch
                time=time+1;

                sub_task2=0;
 }

sub_task3++;
        if(sub_task3>=time_task3)
                                {
                                // This segment is executed every 20 ms
                                if(pcart_cm>35){end_flag=0;return;}
                                if(pcart_cm<-35){end_flag=0;return;}
                                sub_task3=0;
                                 }
T0IR=1; VICVectAddr=0;
}
```

```
void mul_mat(float *p1,float *p2,int row1,int col1,int row2, int col2)
{

int i,j,k;
float M1[no_max][no_max];
float M2[no_max][no_max];
float prod=0;

for(i=0;i<row1;i++)
        {
        for(j=0;j<col1;j++)
                {
                M1[i][j]=*p1;
                p1++;
                }
         }

for(i=0;i<row2;i++)
        {
        for(j=0;j<col2;j++)
                {
                M2[i][j]=*p2;
                p2++;
                }
         }


for(i=0;i<row1;i++)
        {
        for(k=0;k<col2;k++)
                        {
                        prod=0;
                        for(j=0;j<col1;j++)
                                {
                                prod=prod+M1[i][j]*M2[j][k];
                                }AplusB[i][k]=prod;
                        }
        }

}
```

APPENDIX B

Paper: **Meeting real-time constraints using “Sandwich Delays”**

Meeting real-time constraints using "Sandwich Delays"

Michael J. Pont, Susan Kurian and Ricardo Bautista-Quintero
Embedded Systems Laboratory, University of Leicester,
University Road, LEICESTER LE1 7RH, UK.
M.Pont@le.ac.uk; sk183@le.ac.uk; rb169@le.ac.uk
http://www.le.ac.uk/eg/embedded/

Abstract

This short paper is concerned with the use of patterns to support the development of software for reliable, resource-constrained, embedded systems. The paper introduces one new pattern (SANDWICH DELAY) and describes one possible implementation of this pattern for use with a popular family of ARM-based microcontrollers.

Introduction

In this paper, we are concerned with the development of embedded systems for which there are two (sometimes conflicting) constraints. First, we wish to implement the design using a low-cost microcontroller, which has – compared to a desktop computer – very limited memory and CPU performance. Second, we wish to produce a system with extremely predictable timing behaviour.

To support the development of this type of software, we have previously described a "language" consisting of more than seventy patterns (e.g. see Pont, 2001). Work began on these patterns in 1996, and they have since been used it in a range of industrial systems and numerous university research projects (e.g. see Pont, 2003; Pont and Banner, 2004; Mwelwa et al., 2006; Kurian and Pont, 2006; Kurian and Pont, 2007a; Kurian and Pont, 2007b; Ayavoo et al., in press).

This brief paper describes one new pattern (SANDWICH DELAY) and illustrates – using what we call a "pattern implementation example" (e.g. see Kurian and Pont, 2007b) - one possible implementation of this pattern for use with a popular family of ARM-based microcontrollers.

Acknowledgements

Many thanks to Bob Hanmer, who provided numerous useful suggestions during the shepherding process. We also thank the contributors to our workshop session (Joe Bergen, Frank Buschmann, Neil Harrison, Kevlin Henney, Andy Longshaw, Klaus Marquardt, Didi Schütz and Markus Völter) for further comments on this paper at the conference itself.

SANDWICH DELAY

{pattern}

Context

- You are developing an embedded system.
- Available CPU and / or memory resources are – compared with typical desktop designs – rather limited.
- Your system is based on a time-triggered scheduler rather than a "real-time operating system".
- Your system involves running two or more periodic tasks.
- Predictable timing behaviour is a key design requirement.

Problem

You are running two activities, one after the other. How can we ensure that the interval between the release times of the two activities is known and fixed?

Background

In many embedded applications (such as those involving control or data acquisition) variations in the start times of tasks or functions can have serious implications. Such timing variations are known as "release jitter" (or simply "jitter").

For example, Cottet and David (1999) show that – during data acquisition tasks – jitter rates of 10% or more can introduce errors which are so significant that any subsequent interpretation of the sampled signal may be rendered meaningless. Similarly Jerri discusses the serious impact of jitter on applications such as spectrum analysis and filtering (Jerri, 1997). Also, in control systems, jitter can greatly degrade the performance by varying the sampling period (Torngren, 1998; Mart et al., 2001).

In many embedded systems, we wish to keep the levels of jitter to a minimum.

Solution

A SANDWICH DELAY can be used to solve this type of problem. More specifically, a SANDWICH DELAY provides a simple but highly effective means of ensuring that a particular piece of code *always takes the same period of time to execute*: this is done using two timer operations to "sandwich" the activity you need to perform.

To illustrate one possible application of a SANDWICH DELAY, suppose that we have a system executing two functions periodically, as outlined in Listing 1.

```
// Interrupt Service Routine (ISR) invoked by timer overflow every 10 ms
void Timer_ISR(void)
   {
   Do_X();  // WCET* approx. 4.0 ms
   Do_Y();  // WCET approx. 4.0 ms
   }
```

Listing 1: Using a timer ISR to execute two periodic functions.

According to the code in Listing 1, function `Do_X()` will be executed every 10 ms. Similarly, function `Do_Y()` will be executed every 10 ms, after `Do_X()` completes. For many resource-constrained applications (for example, control systems) this architecture may be appropriate. However, in some cases, the risk of jitter in the start times of function `Do_Y()` may cause problems. Such jitter will arise if there is any variation in the duration of function `Do_X()`. In Figure 1, the jitter will be reflected in differences between the values of *ty1* and *ty2* (for example).

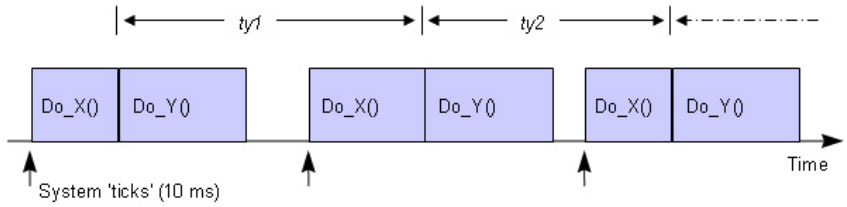

Figure 1: The impact of variations in the duration of `Do_X()` on the release jitter of `Do_Y()`.

We can use a SANDWICH DELAY to solve this problem: please refer to Listing 2.

```
// ISR invoked by timer overflow every 10 ms
void Timer_ISR(void)
   {
   // Execute Do_X() in a 'Sandwich Delay' - BEGIN
   Set_Sandwich_Timer_Overflow(5);       // Set timer to overflow after 5 ms
   Do_X();                               // Execute Do_X - WCET approx. 4 ms
   Wait_For_Sandwich_Timer_Overflow();   // Wait for timer to overflow
   // Execute Do_X() in a 'Sandwich Delay' - END

   Do_Y();  // WCET approx. 4.0 ms
   }
```

Listing 2: Employing a SANDWICH DELAY to reduce release in function `Do_Y()`.

In Listing 2, we set a timer to overflow after 5 ms (a period slightly longer than the worst-case execution time of `Do_X()`). We then start this timer before we run the function and – after the function is complete – we wait for the timer to reach the 5 ms value. In this way, we ensure that – as long as `Do_X()` does not exceed a duration of 5 ms - `Do_Y()` runs with very little jitter†.

Figure 2 shows the tick graph from this example, with the SANDWICH DELAY included.

---

* WCET = Worst-Case Execution Time. If we run the task an infinite number of times and measure how long it takes to complete, the WCET will be the longest execution time which we measure.

† In general, it is not possible to remove all jitter using this approach: we explain why under the heading "Reliability and safety implications".

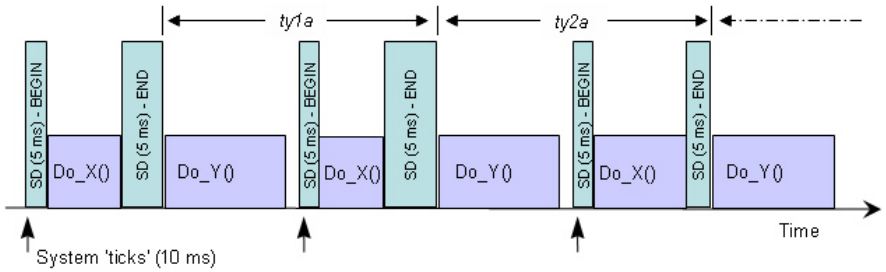

Figure 2: Reducing the impact of variations in the duration of `Do_X()` on the release jitter of `Do_Y()` through use of a SANDWICH DELAY.

Related patterns and alternative solutions

In some cases, you can avoid the use of a SANDWICH DELAY altogether, by altering the system tick interval. For example, if we look again at our `Do_X()` / `Do_Y()` system, the two tasks have the same duration. In this case, we would be better to reduce the tick interval to 5 ms and run the tasks in alternating time slots (Figure 3).

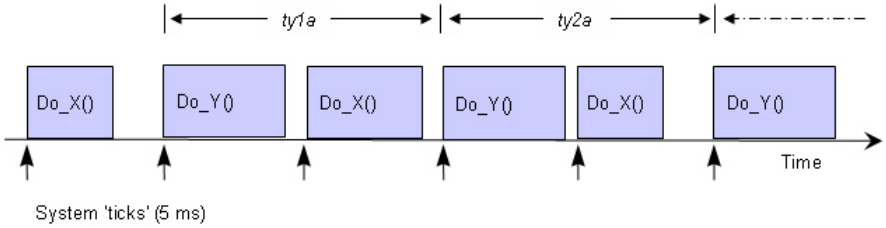

Figure 3: Avoiding the use of SANDWICH DELAYS through changes in the scheduler tick interval.

Please note that this solution will only work (in general) if the tasks in your system have similar durations. Where the tasks do not have the same duration, a scheduler involving multiple timer interrupts may be more appropriate: such a solution is beyond the scope of this paper but is described in detail elsewhere (Nahas and Pont, submitted).

Reliability and safety implications

Use of a SANDWICH DELAY is generally straightforward, but there are three potential issues of which you should be aware.

First, you need to know the duration (WCET) of the function(s) to be sandwiched. If you underestimate this value, the timer will already have reached its overflow value when your function(s) complete, and the level of jitter will not be reduced (indeed, the SANDWICH DELAY is likely to slightly increase the jitter in this case).

Second, you must check the code carefully, because the "wait" function may never terminate if the timer is incorrectly set up. In these circumstances a watchdog timer (e.g. see Pont, 2001; Pont and Ong, 2003) or a "task guardian" (see Hughes and Pont, 2004) may help to rescue your system, but relying on such mechanisms to deal with poor design or inadequate testing is – of course - never a good idea.

Third, you will rarely manage to remove all jitter using such an approach, because the system cannot react instantly when the timer reaches its maximum value (at the machine-code level, the code used to poll the timer flag is more complex than it may appear, and the time taken to react to the flag change will vary slightly). A useful rule of thumb is that jitter levels of around 1 µs will still be seen using a SANDWICH DELAY.

Overall strengths and weaknesses

☺ A simple way of ensuring that the WCET of a block of code is highly predictable.

☹ Requires (non-exclusive) access to a timer.

☹ Will only rarely provide a "jitter free" solution: variations in code duration of around 1 µs are representative.

Example: Application of Dynamic Voltage Scaling

As we note in "Context", we are concerned in this pattern with the development of software for embedded systems in which (i) the developer must adhere to severe resource constraints, and (ii) there is a need for highly predictable system behaviour. With many mobile designs (for example, mobile medical equipment) we also need to minimise power consumption in order to maximise battery life.

To meet all three constraints, it is sometimes possible to use a system architecture which combines time-triggered co-operative (TTC) task scheduling with a power-reduction technique know as "dynamic voltage scaling" (DVS). To achieve this, use of a SANDWICH DELAY is a crucial part of the implementation (and is used to ensure that the complex DVS operations do not introduce task jitter).

The use of SANDWICH DELAYS in this context is described in detail by Phatrapornnant and Pont (2006).

SANDWICH DELAY (C, LPC2000)

{pattern implementation example}*

Context

- You wish to implement a SANDWICH DELAY [this paper]
- Your chosen implementation language is C†.
- Your chosen implementation platform is the NXP‡ LPC2000 family of (ARM7-based) microcontrollers.

Problem

How can you implement a SANDWICH DELAY for the NXP LPC2000 family of microcontrollers?

Background

As with all widely-used microcontrollers, the LPC2000 devices have on-chip timers which are directly accessible by the programmer. More specifically, all members of this family have two 32-bit timers, known as Timer 0 and Timer 1. These can each be set to take actions (such as setting a flag) when a particular time period has elapsed.

In the simplest case, these timers (and other peripheral devices) will be driven by the "peripheral clock" (plck) which - by default - runs at 25% of the rate of the system oscillator (Figure 4).

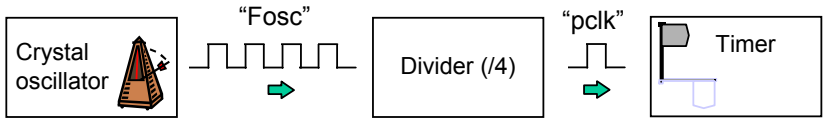

Figure 4: The link between oscillator frequency and timer updates in the LPC2000 devices (default situation).

By taking into account the link between the oscillator frequency and the timer hardware, the timers can be configured so that (for example) a flag is set after a period of 10ms has elapsed. The resulting delay code can be made highly portable.

---

* As the name might suggest, PIEs are intended to illustrate how a particular pattern can be implemented. This is important (in the embedded systems field) because there are great differences in system environments, caused by variations in the hardware platform (e.g. 8-bit, 16-bit, 32-bit, 64-bit), and programming language (e.g. assembly language, C, C++). The possible implementations are not sufficiently different to be classified as distinct patterns: however, they do contain useful information. We say more about PIEs in another paper at EuroPLoP 2006 (Kurian and Pont, in press b).

† The examples in the pattern were created using the GNU C compiler, hosted in a Keil uVision 3 IDE.

‡ Formerly Philips Semiconductors.

Both Timer 0 and Timer 1 are 32-bit timers, which are preceded by a 32-bit pre-scalar. The pre-scalar is in turn driven by the peripheral clock. This is an extremely flexible combination. As an example, suppose that we wished to generate the longest possible delay using Timer 0 on an LPC2100 device with a 12 MHz oscillator. The delay would be generated as follows:

- Both the pre-scalar and the timer itself begin with a count of 0.
- The pre-scalar would be set to trigger at its maximum value: this is 2^{32}-1 (=4294967295). With a 12 MHz oscillator (and the default divider of 4), the pre-scalar would take approximately 1432 seconds to reach this value. It would then be reset, and begin counting again.
- When the pre-scalar reached 1432 seconds, Timer 0 would be incremented by 1. To reach its full count (4294967295) would take approximately 200,000 years.

Clearly, this length of delay will not be required in most applications! However, very precise delays (for example, an hour, a day – even a week) can be created using this flexible hardware.

As a more detailed example, suppose that we have a 12 MHz oscillator (again with default divider of 4) and we wish to generate a delay of 1 second. We can omit the prescalar, and simply set the match register on Timer 1 to count to to the required value (3,000,000 – 1).

We can achieve this using the code shown in Listing 3.

```
// Prescale is 0 (in effect, prescalar not used)
T1PC = 0;

// Set the “Timer Counter Register” for this timer.
// In this register, Bit 0 is the “Counter Enable” bit.
// When 1, the Timer Counter and Prescale Counter are enabled for counting.
// When 0, the counters are disabled.
T1TCR &= ~0x01;   // Stop the timer by clearing Bit 0

// There are three match registetrs (MR0, MR1, MR2) for each timer.
// The match register values are continuously compared to the Timer Counter value.
// When the two values are equal, actions can be triggered automatically
T1MR0 = 2999999;  // Set the match register (MR0) to required value

// When the match register detects a match, we can choose to:
// Generate an interrupt (not used here),
// Reset the Timer Counter and / or
// Stop the timer.
// These actions are controlled by the settings in the MCR register.
// Here we set a flag on match (no interrupt), reset the count and stop the timer.
T1MCR = 0x07;     // Set flag on match, reset count and stop timer

T1TCR |= 0x01;    // Start the timer

// Wait for timer to reach count (at which point the IR flag will be set)
while ((T1IR & 0x0001) == 0)
   {
   ;
   }

// Reset the timer flag (by writing "1")
T1IR |= 0x01;
```

Listing 3: Configuring Timer1 in the LPC2000 family. See text for details.

Solution

A code example illustrating the implementation of a SANDWICH DELAY for an LPC2000 device is given in Listing 4 and Listing 5.

```
/*-------------------------------------------------------------------*-

   Main.C (v1.00)

  ------------------------------------------------------------------

   Simple "Sandwich Delay" demo for NXP LPC2000 devices.

-*-------------------------------------------------------------------*/

#include "main.h"

#include "system_init.h"

#include "led_flash.h"
#include "random_loop_delay.h"

#include "sandwich_delay_t1.h"

/*-------------------------------------------------------------------*-

  int main (void)

-*-------------------------------------------------------------------*/
int main(void)
   {
   // Set up PLL, VPB divider, MAM and interrupt mapping
   System_Init();

   // Prepare to flash LED
   LED_FLASH_Init();

   // Prepare for "random" delays
   RANDOM_LOOP_DELAY_Init();

   while(1)
      {
      // Set up Timer 1 for 1-second sandwich delay
      SANDWICH_DELAY_T1_Start(1000);

      // Change the LED state (OFF to ON, or vice versa)
      LED_FLASH_Change_State();

      // "Random" delay
      // (Represents function with variable execution time)
      RANDOM_LOOP_DELAY_Wait();

      // Wait for the timer to reach the required value
      SANDWICH_DELAY_T1_Wait();
      }


   return 1;
   }

/*-------------------------------------------------------------------*-
  ---- END OF FILE ------------------------------------------------
-*-------------------------------------------------------------------*/
```

Listing 4: Implementing a SANDWICH DELAY for the LPC2000 family (`main.c`)

```
/*-------------------------------------------------------------------*-

   sandwich_delay_t1.c (v1.00)

  ------------------------------------------------------------------

   "Sandwich delay" for the LPC2000 family using Timer 1.


-*-------------------------------------------------------------------*/

#include "main.h"

/*-------------------------------------------------------------------*-

  SANDWICH_DELAY_T1_Start()

  Parameter is - roughly - delay in milliseconds.

  Uses T1 for delay (Timer 0 often used for scheduler)

-*-------------------------------------------------------------------*/
void SANDWICH_DELAY_T1_Start(const unsigned int DELAY_MS)
   {
   T1PC = 0x00;    // Prescale is 0
   T1TCR &= ~0x01; // Stop timer

   // Set the match register (MR0) to required value
   T1MR0 = ((PCLK / 1000U) * DELAY_MS) - 1;

   // Set flag on match, reset count and stop timer
   T1MCR = 0x07;

   T1TCR |= 0x01;   // Start timer
   }

/*-------------------------------------------------------------------*-

  SANDWICH_DELAY_T1_Wait()
  Waits (indefinitely) for Sandwich Delay to complete.

-*-------------------------------------------------------------------*/

void SANDWICH_DELAY_T1_Wait(void)
   {
   // Wait for timer to reach count
   while ((T1IR & 0x01) == 0)
     {
     ;
     }

   // Reset flag (by writing "1")
   T1IR |= 0x01;
   }

/*-------------------------------------------------------------------*-
  ---- END OF FILE ------------------------------------------------
-*-------------------------------------------------------------------*/
```

Listing 5: Implementing a SANDWICH DELAY for the LPC2000 family (example): file (`sandwich_delay_t1.c`)

References

Ayavoo, D., Pont, M.J., Short, M. and Parker, S. (in press) "Two novel shared-clock scheduling algorithms for use with CAN-based distributed systems", to be published in: *Microprocessors and Microsystems*.

Cottet, F. and David, L. (1999), "A solution to the time jitter removal in deadline based scheduling of real-time applications", 5th IEEE Real-Time Technology and Applications Symposium - WIP, Vancouver, Canada, pp. 33-38.

Furber, S. (2000) "*ARM System-on-Chip Architecture",* Addison-Wesley.

Jerri, A. J. (1997), "The Shannon sampling theorem: its various extensions and applications a tutorial review", Proc. of the IEEE, Vol. 65, pp. 1565-1596.

Hughes, Z.H. and Pont, M.J. (2004) "Design and test of a task guardian for use in TTCS embedded systems". In: Koelmans, A., Bystrov, A. and Pont, M.J. (Eds.) Proceedings of the UK Embedded Forum 2004 (Birmingham, UK, October 2004), pp.16-25. Published by University of Newcastle upon Tyne [ISBN: 0-7017-0180-3].

Key, S.A., Pont, M.J. and Edwards, S. (2004) "Implementing low-cost TTCS systems using assembly language". Proceedings of the Eighth European conference on Pattern Languages of Programs (EuroPLoP 2003), Germany, June 2003: pp.667-690. Published by Universitätsverlag Konstanz. ISBN 3-87940-788-6.

Kurian, S. and Pont, M.J. (2006) "Evaluating and improving pattern-based software designs for resource-constrained embedded systems". In: C. Guedes Soares & E. Zio (Eds), "Safety and Reliability for Managing Risk: Proceedings of the 15th European Safety and Reliabilty Conference (ESREL 2006), Estoril, Portugal, 18-22 September 2006", Vol. 2, pp.1417-1423. Published by Taylor and Francis, London. ISBN: 0-415-41620-5 (for complete 3-volume set of proceedings). ISBN: 978-0-415-42314-4 (for Volume 2).

Kurian, S. and Pont, M.J. (2007a) "Maintenance and evolution of resource-constrained embedded systems created using design patterns", *Journal of Systems and Software*, **80**(1): 32-41.

Kurian, S. and Pont, M.J. (2007b) "Restructuring a pattern language which supports time-triggered co-operative software architectures in resource-constrained embedded systems". In: Proceedings of the Eleventh European conference on Pattern Languages of Programs (EuroPLoP), held in Germany, July 2006 [This conference].

Mart, P., Fuertes, J. M., Ramamritham, K. and Fohler, G. (2001), "Jitter Compensation for Real-Time Control Systems", 22nd IEEE Real-Time Systems Symposium (RTSS'01), London, England, pp. 39-48.

Mwelwa, C., Athaide, K., Mearns, D., Pont, M.J. and Ward, D. (2006) "Rapid software development for reliable embedded systems using a pattern-based code generation tool". Paper presented at the Society of Automotive Engineers (SAE) World Congress, Detroit, Michigan, USA, April 2006. SAE document number: 2006-01-1457. Appears in: Society of Automotive Engineers (Ed.) "In-vehicle software and hardware systems", Published by Society of Automotive Engineers. [ISBN: 0-7680-1763-7].

Nahas, M. and Pont, M.J. (submitted) "The impact of scheduler implementation on task jitter in real-time embedded systems".

Phatrapornnant, T. and Pont, M.J. (2006) "Reducing jitter in embedded systems employing a time-triggered software architecture and dynamic voltage scaling" IEEE Transactions on Computers, **55** (2): 113-124.

Philips (2004) "LPC2119 / 2129 / 2194 / 2292 / 2294 User Manual", Philips Semiconductors, 3 February, 2004.

Pont, M.J. (2001) “Patterns for Time-Triggered Embedded Systems: Building Reliable Applications with the 8051 Family of Microcontrollers”, Addison-Wesley / ACM Press. ISBN: 0-201-331381.

Pont, M.J. (2003) “Supporting the development of time-triggered co-operatively scheduled (TTCS) embedded software using design patterns”, Informatica, 27: 81-88.

Pont, M.J. and Banner, M.P. (2004) “Designing embedded systems using patterns: A case study”, *Journal of Systems and Software*, 71(3): 201-213.

Pont, M.J. and Ong, H.L.R. (2003) “Using watchdog timers to improve the reliability of TTCS embedded systems”, in Hruby, P. and Soressen, K. E. [Eds.] *Proceedings of the First Nordic Conference on Pattern Languages of Programs, September, 2002 (“VikingPloP 2002”)*, pp.159-200. Published by Microsoft Business Solutions. ISBN: 87-7849-769-8.

Pont, M.J., Norman, A.J., Mwelwa, C. and Edwards, T. (2004) “Prototyping time-triggered embedded systems using PC hardware”. Proceedings of the Eighth European conference on Pattern Languages of Programs (EuroPLoP 2003), Germany, June 2003: pp.691-716. Published by Universitätsverlag Konstanz. ISBN 3-87940-788-6.

Torngren, M. (1998), "Fundamentals of implementing real-time control applications in distributed computer systems", Real-Time Systems, 14, pp. 219-250.