Available online at www.sciencedirect.com
SCIENCE DIRECT®
Microprocessors and Microsystems 27 (2003) 159–169
MICROPROCESSORS AND MICROSYSTEMS
www.elsevier.com/locate/micpro

Establishing timing requirements for control loops in real-time systems

Iain Bate*, John McDermid, Peter Nightingale

Department of Computer Science, University of York, York YO10 5DD, UK

Received 23 May 2002; revised 20 November 2002; accepted 17 December 2002

Abstract

Advances in scheduling theory have given designers of control systems greater flexibility over their choice of timing requirements. This could lead to systems becoming more responsive and more maintainable. However, experience has shown that engineers find it difficult to exploit these advantages due to the difficulty in determining the 'real' timing requirements of systems and therefore the techniques have delivered less benefit than expected. Part of the reason for this is that the models used by engineers when developing systems do not allow for emergent properties such as timing. The paper presents an approach and framework for addressing the problem of identifying an appropriate and valid set of timing requirements in order that the best use can be made of the advances in scheduling theory by the use of modelling techniques that allow for emergent properties such as timing behaviour.

Keywords: Real-time systems; Control systems; Scheduling; Model based development

1. Introduction

This paper addresses the perennial problem of how to identify an appropriate and valid set of timing requirements for a hard real-time system. Our main concern is with systems where failure(s) to meet timing requirements can lead to dangerous behaviour, and may ultimately lead to loss of life. Over the years, research on real-time systems has evolved techniques which provide greater flexibility in scheduling whilst still providing a means for guaranteeing that timing requirements are met [1,2]. The increased flexibility was expected to give many benefits, including more efficient use of resources and simpler maintenance of schedules when changes to the control software are made. Maintaining schedules is often a costly and error prone manual process, therefore these techniques have the potential to offer significant economic benefits as well as engineering benefits.

Experience has shown that engineers find it difficult to exploit this increased flexibility and hence the techniques have delivered less benefit than expected. Based on our own experience and that of others in industry [3,14], the main cause of the shortfall in benefit gained is an absence of information about the *true* timing requirements which is needed to make best use of the approaches. In many cases current systems are developed with simple timing requirements, such as a timing margin to be achieved. In other cases the timing requirements are largely historic, and are simply expressed in terms of iteration rates, which have been proven effective in previous designs. Despite the changing contexts between systems, this strategy is normally successful because the requirements are over conservative, e.g. update rates specified are much faster than needed. Even where more modern control law design environments (e.g. Matlab/Simulink [4]) are used the control models are often produced assuming a particular computational model. For example a 50 ms cycle/20 Hz bandwidth is chosen because there is a regular clock tick in the system with a period of 25 ms (i.e. 40 Hz) and therefore it is easier to release tasks at a harmonic of this frequency.

An often-used way to select timing requirements is the Nyquist criterion [6]. Nyquist's criterion places a lower bound on the sampling period—equal to twice the highest input frequency. This highest frequency can be open ended, for example a square wave has frequency components up to infinity so leaving an unanswered question of where would we draw the line. Similarly while a plant model may be limited to order N, the actual plant would be much more complicated. This could result in a sampling rate that is too

* Corresponding author.
E-mail addresses: iain.bate@cs.york.ac.uk (I. Bate), john.mcdermid@cs.york.ac.uk (J. McDermid), pwn101@cs.york.ac.uk (P. Nightingale).

0141-933/03/$ - see front matter © 2003 Elsevier Science B.V. All rights reserved.
doi:10.1016/S0141-9331(03)00015-2

high. Therefore practitioners normally assume the input signal can be represented by a frequency range within which *N%* of the intensity exists. Also, Nyquist's criterion is based on the assumption that the system under control is time invariant which in the case of a computer controlled system is not valid. Despite these assumptions, the Nyquist's criterion does not lead to instability as long as the time variations in when sampling occurs does not cause the bound on the sampling rate derived using Nyquist's criterion to be exceeded.

The work contained in this paper does not purpose an alternative to the Nyquist's criterion but instead a complementary method. Even with Nyquist's criterion, there is a need to choose an actual sampling rate since the criterion only provides a lower bound on the rate.

A major contributor to the methods using for selecting timing requirements is that the research and into the practical use of control theory and scheduling theory have largely been carried out in isolation [5]. Thus, for example, work on advanced control regimes such as H∞ [6], which might benefit from more sophisticated scheduling has not been the subject of joint work between control engineers and real-time systems researchers. Perhaps as a consequence there are few examples of H∞ being used in real-time embedded control systems.

The paper presents an approach and framework for addressing the problem of identifying an appropriate and valid set of timing requirements in order that the best use can be made of the advances in scheduling theory. The essence of the approach is to use component-based models that allow for emergent properties of systems, in this case timing, so that the models are more representative of how a real system would actually behave. Then, heuristic search techniques are used to explore the design space and to identify timing requirements which enable properties such as control stability to be achieved, thus deriving and validating the requirements against more fundamental properties of the control system. The advantage of using heuristic search techniques instead of traditional model-based design approaches, such as root locus [6], include it allows many properties and effects to be considered at the same time and their demands on the system to be traded-off against one another.

The work presented here is intended for use in a range of control problems, but is illustrated with the Proportional Integral Differential (PID) control approach [6].

The rest of the paper is structured as follows. Section 2 gives further background on the control techniques to be used in the context of this work. It also provides a technical motivation (as opposed to the 'economic' motivation outlined above) for seeking a systematic approach to deriving timing requirements. Section 3 presents the approach outlining the heuristics used, and the costs of evaluating the requirements. Section 4 contains two simple case studies which have been used to evaluate the approach, as well as presenting a discussion of how the resulting timing requirements may be used. Finally, Section 5 gives a summary and suggests possible future developments for the work.

2. Background and motivation

All scheduling approaches require a minimum set of information about timing requirements so that an appropriate scheduler can be produced. For most scheduling approaches the minimum set of information for any approach is the deadline and period of tasks [7,8]. This section explains why these requirements are important in the context of PID loops and how they can be generated by considering basic control properties.

2.1. PID loop

The principal purpose of a PID loop is to ensure the controller meets its objectives. Objectives of the controller could include responsiveness to input, stability, accuracy and limits on data are maintained. Fig. 1 depicts a typical PID loop, in a control system, being used to control the operation of a plant. The Figure shows the key aspects and components of the controller. There is only one input to, and one output from, the control system. The output of the control system is the plant input. The control system input is the difference between the input demand (typically from the operator of the plant) and the plant's actual output, and it is referred to as the error (in this paper error is defined as the difference between actual and desired plant state).

In the computer-based approach, the *Input Demand* and the *Actual Plant Output* are usually analogue signals. The computer performs the rest of the processing in the digital domain. Converters are used to sample the analogue signals, e.g. to produce the *Error* input, and then converted back to analogue values at the output. Converting back to an analogue signal is often referred to as digital to analogue conversion, or de-sampling.

The controller (PID loop) works by adding scaled versions of the error, differential of the error and integral of the error to achieve the desired control response. The scale factors are K_P for the proportional term, K_I for the integral term, and K_D for the differential term.

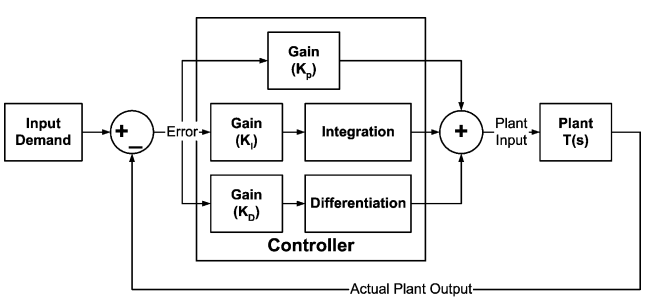

Fig. 1. Typical PID loop.

The integration term is used to control steady state error (higher values of K_I reduces steady errors) but too high a value can lead to instability. Assuming the value of the differentiation term (K_D) is not too high, it improves the speed of closed loop response and increases the stability margin. The 'design problem' at this level is one of choosing the gains (K_P, K_I, K_D) to achieve (approximately) the desired control response.

In industrial practice it is common for a controller to be developed as a continuous system based on the system's response in the frequency domain. Often modelling packages or special purpose plant simulations are used to validate the requirements. If a computer-based implementation is to be used, then once the requirements have been established in the continuous domain they are converted to the discrete domain. Typically the conversion involves calculating the PID loop gains (K_P, K_I, K_D) based on the assumption that a constant sampling period is used. This means the conversion is performed based on an idealised model of the computer system. In other words the conversion uses unrealistic assumptions, e.g. infinite processing bandwidth and zero jitter in sampling the inputs (jitter is the variation in time when an action occurs between one cycle of the controller and the next). Our approach addresses this shortcoming by taking into account the constraints of real computer systems, and thus enables valid and realistic requirements to be produced. To explain how this is done the rest of this section explains in more detail the relationship between computational properties such as jitter and control properties such as stability.

2.2. Scheduling properties

It is, of course, essential that the sampling, core functions and de-sampling tasks are executed in that order. Other work, e.g. [7,9], has shown how to specify and control the precedence of functionality for a PID loop to ensure that these requirements are met. More importantly for our discussions in this paper, sampling and de-sampling will be subject to jitter due to limits on the accuracy of clocks, and the interference of other software running on the processor. Thus the true sampling times will vary, and the simple assumptions of fixed and precise iteration rates used in validating the control model will not be representative of the computations, which occur in practice.

Fig. 2 presents properties for a typical transaction of a control loop that can be controlled by the scheduler. The three tasks are sensor capture, calculation and actuation output. The figure shows:

- how each task has jitter comprising both release and execution jitter as well as an invariant in its execution time
- there is jitter on both sensor capture (referred to as sampling jitter) and actuation (referred to as desampling jitter),
- a task must be completed before the next task in the transaction starts its execution so that the next task can use fresh data,
- the response time of a transaction is equal to the time between the release of the first task and the completion of the last task (the worst-case response time for a transaction must be less than its deadline), and
- the period of a task is the time between two consecutive earliest releases.

Fig. 3 shows the earliest and latest release, earliest start of execution, and earliest and latest completion for an individual task. The following are a number of properties related to the diagram.

1. Each task has a variation in execution time in the range [BCET, WCET].
2. Each task suffers from release jitter.
3. Each task has a deadline (denoted by D) greater than its WCET.

2.3. Control and scheduling interactions

The classic definition of stability in general terms is that a system is stable if bounded inputs return outputs that remain bounded for all time [6]. Based on this definition,

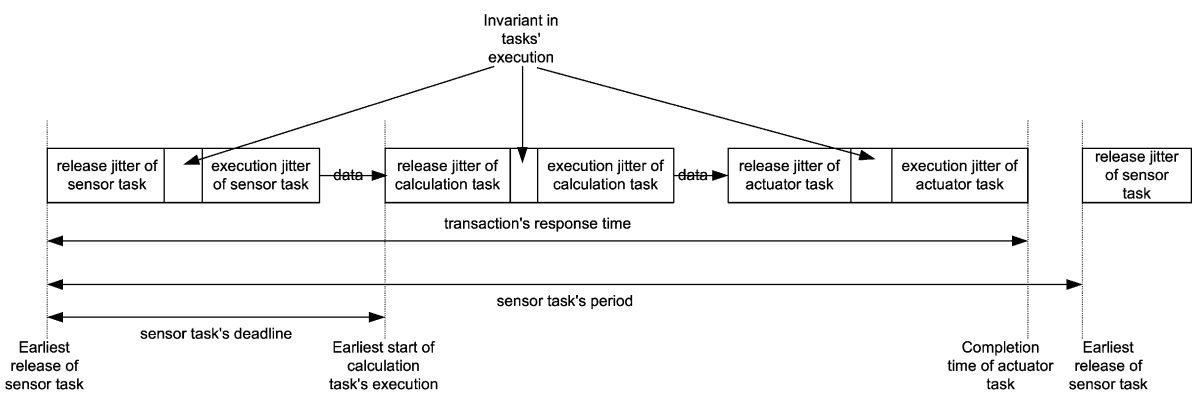

Fig. 2. Scheduling properties for a transaction.

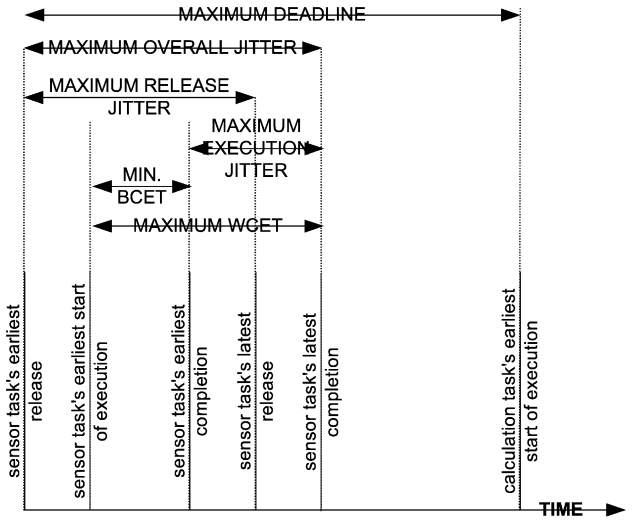

Fig. 3. Scheduling properties for a task.

the stability of a control system can be judged on its responsiveness (time taken to settle to $N\%$ of the intended value) and its degree of overshoot (the amount by which a value exceeds that intended). There are three basic phenomena that can lead to a system being unstable. These are:

- slow and irregular responses to stimuli caused by finite update rates and jitter;
- errors within calculations, e.g. due to sampling inputs at the 'wrong' time;
- the gain at the crossover frequency being greater than unity.

The stability of the system is related to the frequency response of the control system (and hence the calibration settings for the control loop) as well as the other two properties discussed. From a scheduling perspective, only the responsiveness and size of the errors can be controlled since the gain is a functional property of the control software. The following subsections address these issues as well as a discussion of how these relate to timing requirements.

2.3.1. Responsiveness

The responsiveness of the system is dependent on two factors: the degree of damping that the PID loop exerts on the input parameters and the sampling rate. The first of these is a control issue and the second a scheduling issue. With respect to the sampling rate, too fast a rate uses valuable resources unnecessarily but a slower rate increases the time before the PID loop receives changes in the Input Demand and hence increases the response time. Therefore a trade-off needs to be made between the sampling rate, responsiveness and resource availability for other tasks.

2.3.2. Error

The error in the system is dependent on two factors: the time to respond to changes in input stimuli (as previously discussed in Section 2.3.1), and effects due to sampling jitter and end-to-end response time. The end-to-end response time of the control loop introduces error because the output at any time should relate to the input at that time, therefore latency introduces error. The sampling jitter introduces errors due to the differentiation and integration calculations assuming a constant period. The de-sampling jitter introduces error because in the feedback loop (and subsequently the control loop) there is variation when the exact output is made so it introduces error in the calculations of the differentiation and integration of the difference between the input and output signals.

An example of how errors originate is given by showing how input jitter affects the differentiation function of the PID loop. A differentiation is normally approximated using Eq. (1). However, since the gain factors for the PID loop are dependent on the sampling period, then Δt is not actually altered in the calculations, and this leads to error. Consider a situation as shown in Eq. (2) where $\delta x/\delta t$ is actually 2. Eq. (3) shows the effect of the previous sample ($x(t-\Delta t)$) being early by half a sampling period. Whereas, Eq. (4) shows the effect of the current sample being taken half a sampling period late. In both examples the error caused by jitter is 50%, this results in a difference between the two of a factor of 3. Thus variations in sampling time caused by limited precision of the system clock, and by interference from other tasks, can cause computational errors, and may lead to instability.

In our experience, interference is a particular problem. For example, even if the tasks related to control are scheduled at much higher rates than strictly needed, dependent on how the system is scheduled and the rest of the system's workload, the amount of jitter can still be significant. Hence control system performance is adversely affected. The timing properties and requirements of 'other' tasks will necessitate trade-offs in the timing requirements of the system. For instance, if the tasks' utilisation of the processing resource is low, then the control tasks can be executed at a higher rate. However, if the utilisation is high, then the control tasks may have to be executed at a slower rate and hence their jitter controlled more carefully. In addition for the high utilisation case, the timing requirements and properties of the 'other' tasks would have to be managed more stringently.

$$\frac{\delta x}{\delta t} \approx \frac{x(t) - x(t-\Delta t)}{\Delta t} \tag{1}$$

$$\frac{\delta x}{\delta t} \approx \frac{x(t) - x(t-\Delta t)}{\Delta t} = 2 \tag{2}$$

$$\frac{\delta x}{\delta t} \approx \frac{x(t) - x(t-0.5\Delta t)}{\Delta t} = 1 \tag{3}$$

$$\frac{\delta x}{\delta t} \approx \frac{x(t + 0.5\Delta T) - x(t - \Delta t)}{\Delta t} = 3 \quad (4)$$

where

t is the current time,
Δt is the sampling period,
$x(t)$ is the value of the variable to be differentiated at time t, and,
$\delta x/\delta t$ is the result of the differentiation.

2.3.3. Summary of interactions

In summary, it is important for reasons of stability and responsiveness that appropriate values of period and deadline are specified and met for the control loop. To be effective, the approach to setting these requirements must allow for the effects of the implementation (where possible independence of the scheduling policy used should be maintained), and must be carried out in conjunction with deriving PID loop gains, otherwise the resulting system may be unstable, or fail to meet its design objectives in some other way. For any given control loop, there are a set of possible timing requirements that can achieve the desired results. These requirements can vary from short periods with deadline equal to period, to longer periods but deadlines being less than their period. Section 3 presents our framework for deriving timing requirements.

3. Modelling approach and framework for evaluating timing requirements

For the purpose of this work, no specific scheduling approach is assumed. Instead, it is assumed that the 'scheduling problem' can be divided into two parts: devising a set of timing requirements and verifying that a chosen schedule meets those requirements. There are several solutions to the latter problem, e.g. [1,7], so, for the purposes of this paper, we consider this to be a solved problem, and focus on the issue of generating requirements.

Our approach seeks to build on the capability of existing tools for developing control laws, and to exploit the power of heuristic search to explore the 'design space' produced by the interplay of task periods, deadlines and jitter as well as the loop gains (K_P, K_I, K_D).

There are three significant parts to the framework that has been developed in MATLAB [4]:

1. Modelling the PID Operation
2. Tuning the PID loop
3. Determining the valid set of timing requirements by stepping through a range of combinations of periods and deadlines and using the previous two stages to decide whether the combination meets the criteria for the control system.

The following subsections discuss each of these parts in more depth.

3.1. Modelling the PID operation

The MATLAB model for simulating a classical PID control loop is shown in Fig. 4. The key elements here are that the gains of the PID loop can be fixed or programmable, and that the timing of the sensor and actuation is controlled by the representative sampling signal produced by a discrete pulse generator. The purpose of the signal is to simulate the varying jitter on when actions (sampling and actuation) occur as shown in Fig. 2. That is, the signal is used to control when signals are converted from the continuous to the discrete domain and vice versa. As such the signal was generated every $T + \delta$ time units, where T is the ideal sampling period, δ is a random value in the range $[0, J_{MAX}]$ and J_{MAX} is the maximum possible jitter according to Eq. (5). The framework was developed so that δ could be chosen using one of two distributions, Normal and Uniform.

$$D = BCET + J_{MAX} \quad (5)$$

The input $I(t)$ to the model is currently a 'pure' signal in that the framework does not automatically add disturbances such as noise. However, it would be straightforward for the signal

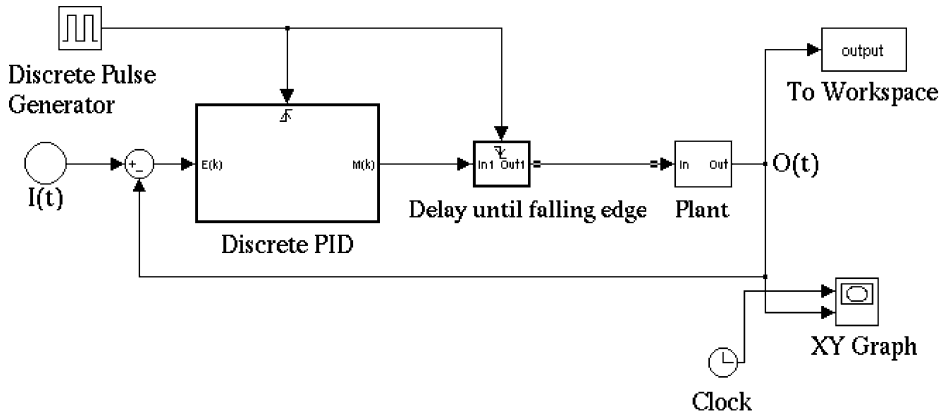

Fig. 4. MATLAB model of the framework.

Table 1
Transformations between continuous and discrete domains

Parameter in continuous domain	Parameter in discrete domain (sampling period = T)
K_P	K_P
K_I	$T \times K_I$
K_D	K_D/T

source to be replaced by one with a noise generator adding to the demand signal. The framework also contains a plant model which can be anything from a 'real' system to a computer-based model as long as it meets our assumptions of a single input and single output.

This simulation model is used to explore the performance of the control system with a given set of parameters, e.g. values for K_P, K_I, K_D, deadlines, etc.

3.2. Tuning the PID loop

It is possible to use PID loop gains in the discrete domain that were originally established in the continuous domain but first they must be transformed using relationships based on the sampling period and the discretisation method. In the context of our approach, the relationships are as shown in Table 1.

However, in the framework produced, an option is to tune the actual gains for any given period and/or deadline. This is where we make use of heuristic approaches, specifically genetic algorithms [10] configured to search the design space for a set of valid and appropriate requirements. Heuristic approaches are generally based on a recursive procedure for mutating (or tuning) a design solution (with the aim of achieving an 'optimum' solution) based on calculations of a fitness function that is used to judge whether the design solution meets the chosen criteria. The basic differences between the types of heuristic approaches are the nature of the fitness functions and the mechanisms used to tune the design solution based on the results of the fitness function calculation.

The advantage of using heuristic search techniques instead of traditional model-based design approaches, such as root locus [6], include that it allows many properties and effects to be considered at the same time and their demands on the system to be traded-off against one another.

Specifically, the tuning is performed using a genetic algorithm whose inputs are:

1. The saturation value for the actuator—i.e. the largest magnitude of output allowed from the controller.
2. The maximum allowed overshoot.
3. The maximum and minimum values of the input demand. This can be used for calculating the maximum controller response allowed on a unit step input as shown in Eq. (6).

$$O_{\max} = \frac{\textit{saturation value}}{I_{\max} - I_{\min}} \tag{6}$$

where
$O_{\max}$ is the maximum allowed controller response based on a unit step input,
$I_{\max}$ is the max value for the step input,
$I_{\min}$ is the min value for the step input.

4. The period of the sensor capture task—i.e. the sampling period. It is assumed that other tasks in the transaction have a rate that is harmonic of sensor capture task's rate.
5. The deadline—used to calculate J_{MAX} refer to Eq. (5) for details.
6. The length of the simulation—the longer this is, the more accurate the calculations are.

Each time the design space is searched (a simulation run) the fitness of the result is assessed, and results are retained if they are 'better' than previous results. This judgement is made according to a fitness function, so-called by analogy with 'survival of the fittest'. There is a substantial literature on design of fitness functions, e.g. Ref. [10]. Our work has built on the standard results, but modified them slightly for our purposes.

The fitness function used is modified from the standard genetic algorithms in that tunings that cause the PID controller to saturate the actuator score zero, i.e. they are marked as sufficiently 'unfit' that they will not survive. In all other cases Eq. (7) is used. This approach prevents the specifics of the optimisation of the chosen fitness function allowing unacceptable cases through. Hence the integrity of the results is assured and the only impact of the chosen fitness function is the selection and prioritisation of the attributes against which quality can be judged [13,15].

In Eq. (7), the GAT term is a penalty for overshooting, proportional to the length of time spent outside the bound. G is a weighting factor that compensates for an individual system's measurement units (e.g. Newtons) and scale of input/output. The value of G is chosen such that GAT is generally an order of magnitude less than $\int t^2|O(t) - I(t)|\mathrm{d}t$. Trails have shown that the timing requirements generated using the framework are relatively insensitive to the exact value. In the examples considered within this paper, a value of 1000 is chosen for G.

It is noted that most control systems utilise anti-windup protection to deal with overshoots by limiting the outputs. However, for the context of this work, the system is to be tuned to keep the outputs within range instead of relying on protection mechanisms and that the mechanisms should only be used to ensure failsafe operation. The integral term is intended to be a measure of how good the response is. The error, $E(t)(= O(t) - I(t))$, becomes rapidly more important

as time increases to reward tunings that settle fast.

$$F = \frac{1}{\int t^2 |O(t) - I(t)| dt + GAT} \quad (7)$$

where units of time are seconds,

F is the output of the fitness function,
T is the sampling period, t is the time in the range [0, simulation length],
$O(t)$ is the observed plant output at time t,
$I(t)$ is the input demand at time t, G is a weighting factor,
A is the number of time periods for which the process variable was above the overshoot bound.

The tuning approach for each combination of deadline and period consists of two steps. Firstly the genetic algorithm mutates the previous set of PID loop gains based on the output of the fitness function. Then, the MATLAB model is exercised with the new PID loop gains. While the MATLAB model is executing, the parameters of interest are returned (e.g. A from Eq. (7)). These parameters of interest can then be used to calculate the fitness function again in the first step. The two steps are recursively followed with a goal of optimising control system performance until the allowed simulation time is reached. The optimisation strategy means that the longer the simulation time the better the results obtained.

During the tuning of the PID loops' gains, the need for appropriate control behaviour (stability, overshoot and responsiveness) is accounted for. Initially, stability was measured using the Routh–Horowitz Criterion [6] by deriving a system model based on the step response and then performing root-locus assessment of where the poles and zeros exist to see if the Criterion is met. However, experimentation showed that responsiveness and overshoot provided a faster and more controllable assessment, and that these parameters relate to whether the system is stable as defined in Section 1.

3.3. Overall evaluation technique

Before the evaluation, the user, via a provided interface, defines the limits for the period of individual activities within the system (e.g. sampling of signals) and the size of steps taken between the limits (e.g. periods between *zero* and *ten* might be evaluated in steps of *one*, resulting in evaluations being performed at 0,1,2,…,9,10). The valid set of timing requirements, as chosen by the fitness function, is then found by systematically searching the possible set of periods ($[T_{min}, T_{max}]$) and deadlines ($[0, T_{current}]$) for values where the tuning algorithm can meet the criteria discussed in Section 3.2. ($T_{current}$ refers to the current period being evaluated.) The searching is performed by stepping through the possible range of periods insteps defined by the user, e.g. 5% steps. Given a set of periods and deadlines, then other requirements can be calculated using the equations defined in Section 2.2. For example, the maximum overall jitter can be calculated using the relationships defined in Eq. (5), and later shown to be met based on its individual components of release jitter and execution jitter using properties (such as execution profiles) from the actual systems. The main input to the system used during the evaluation is a step response, which was chosen because of its wide frequency response.

4. Evaluation

In this paper, two examples are used to evaluate the technique. Each example is chosen because it represents a commonly occurring class of control problem. Firstly, a car cruise control system [11] is used because it normally operates in a stable fashion. Thus it represents a class of systems, e.g. engine control systems, and flight control systems for civil aircraft. Secondly, an inverted pendulum [12] is used because it normally operates in an unstable fashion with stability only being possible with external control assistance. Therefore it represents an important, albeit rather smaller, class of systems, e.g. military fast jet flight control systems.

4.1. Case study 1—cruise control

The plant model (i.e. the car) can be represented by the first-order system depicted in Fig. 5 and mathematically using Eq. (8). For this example, the input, $F(t)$, is the accelerating force in Newtons, and the output, $v(t)$, is the velocity in metres per second. The variable under control is the velocity.

$$\frac{v(s)}{F(s)} = \frac{1}{1000s + 50} \quad (8)$$

Due to the fact the car is an integrating system, it has a slow over-damped response. An example step response from zero to one at time zero is shown in Fig. 6. The input value is a step function of input from zero to one at time zero. The line at output = 1.1 represents the overshoot limit of 10%. There are also lines at 5% above and below the input demand of unity output. Fig. 6 illustrates a settling time of approximately 5 seconds.

The evaluation was performed on a PC running Linux with an Athlon 700 MHz processor using the approach described in Section 3.3. The total evaluation time was approximately 3 hours. (It should be noted that the

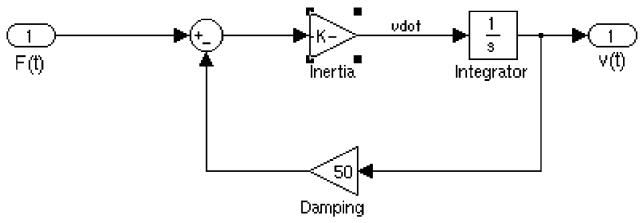

Fig. 5. Model of a car's velocity.

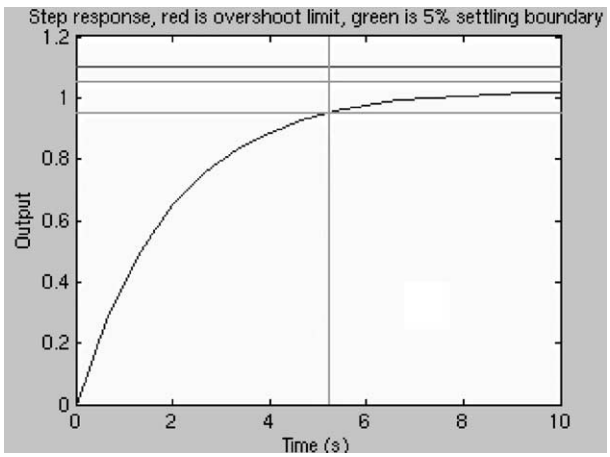

Fig. 6. Step response of the car system.

simulation time can be adjusted as described in Section 3.2, but the longer the simulation time the better the results obtained.) Two graphs (Figs. 7 and 8) were produced that combined the results of all the runs. In all graphs, a dot indicates that the system at that period and deadline passed the suitability test (i.e. test for overshoot and responsiveness limits was met, and the system was stable) and hence represents a valid combination of period, deadline and gains, and a cross indicates that it did not pass the suitability test.

Fig. 7 illustrates the effect of sampling jitter at each of a range of periods with the separation between sample and actuation being held constant at 0.1 seconds i.e. assuming a 100 ms response time for the tasks in the system that execute between the sampling and activities. Fig. 8 illustrates the effect of sampling jitter at each of a range of periods and the effect of variable latency (chosen using a random distribution) between sample and actuation.

The choice of actual requirements will depend on other factors, e.g. an assessment of the practicality of meeting the requirements given the other load on the processor. For

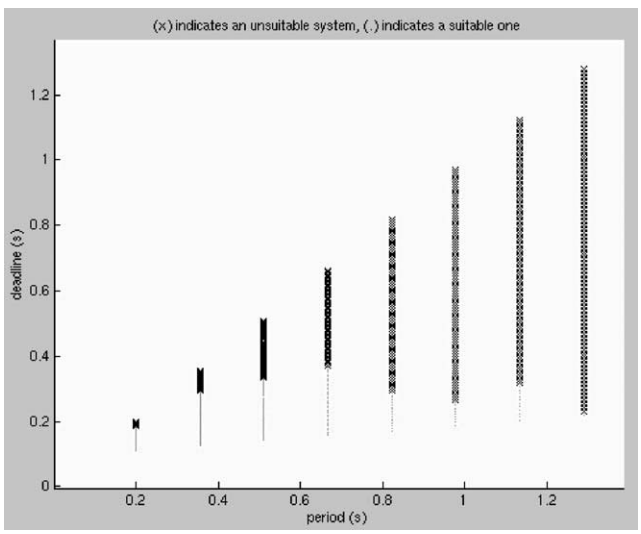

Fig. 7. Deadline versus period with constant separation.

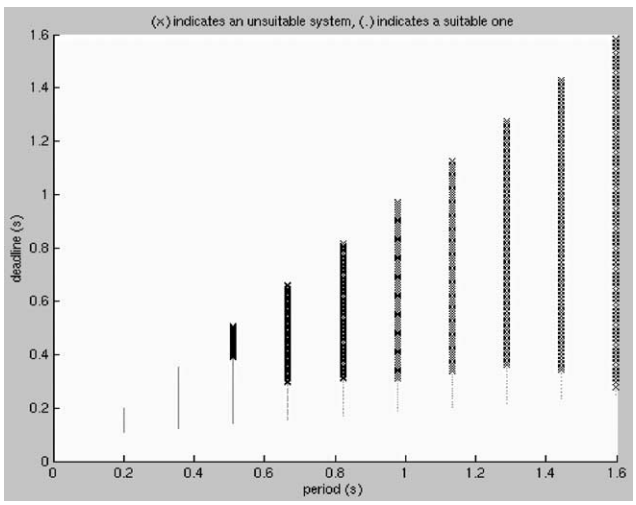

Fig. 8. Deadline versus period with variable separation.

example, a period of 0.5 and a deadline of 0.2 might be chosen because they allow other timing requirements to be met whilst allowing some margin for ‘relaxing’ requirements without causing instability.

Fig. 7, and to a lesser extent Fig. 8, shows the two effects that were anticipated:

1. For periods up to 0.7 seconds, as the period increases the maximum deadline at which the system meets its criteria increases at a similar rate. The increasing maximum deadline is due to the possible range of deadlines increasing (since our evaluation only considers situations where deadline is less than or equal to period), and the relatively small period means errors due to the difference between the actual input and the current sampled value are small and hence the tuner can counteract the effects of errors due to jitter.
2. For periods over 1 seconds, as the period increases the maximum deadline at which the system meets its

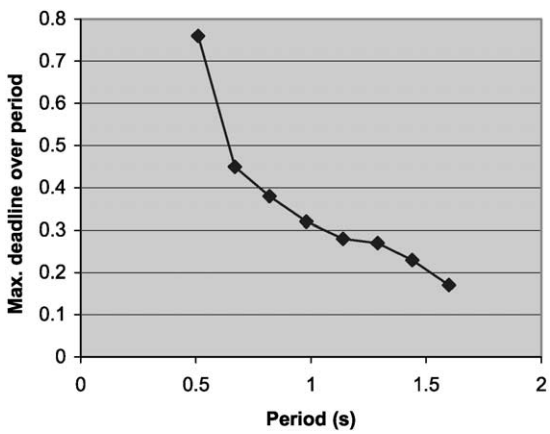

Fig. 9. Tolerated deadline over period versus period.

criteria reduces. The reason is the tuner can only counteract the effects of the increasing difference between actual and sampled input (and the subsequent errors caused) by reducing jitter.

In the range between (i.e. period is greater than 0.7 seconds but less than 1 seconds), the maximum deadline remains fairly steady as the two effects counteract each other.

Fig. 9 further shows the relationship between the ratios of maximum possible deadline with period (y-axis) as the period (x-axis) increases.

4.2. Case study 2—inverted pendulum

The plant model (i.e. inverted pendulum) [12] can be mathematically represented by a third-order system using Eq. (9). The controller output represents the speed of the cart (in m/s), $v(t)$, and the variable under control is the angle of the pendulum from the vertical, $\theta(t)$. The inverted pendulum is unstable without control, and as such is more difficult than the car example to control. The pendulum is made of a stiff rod and a weight, balanced, inverted, on a cart as illustrated in Fig. 10.

$$\frac{\theta(s)}{v(s)} = \frac{4.546s}{s^3 + 0.182s^2 - 31.182s - 4.454} \tag{9}$$

The transfer function model being used for the inverted pendulum has been simplified so that it can be represented as a linear equation. It therefore neglects non-linear effects such as the weight hitting the surface that the cart is running on, and it has been optimised for situations where the pendulum does not travel more than a few degrees away from the vertical. Fig. 11 shows the step response for this pendulum system when controlled by a suitably tuned PID controller. The period of the controller was set very small for this tuning. The tuning was performed with a 50% overshoot limit, and the settling boundary of 5% either side of unity output is shown on the graph.

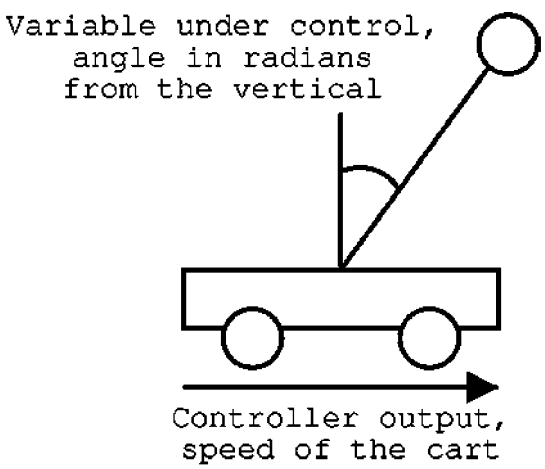

Fig. 10. Inverted pendulum.

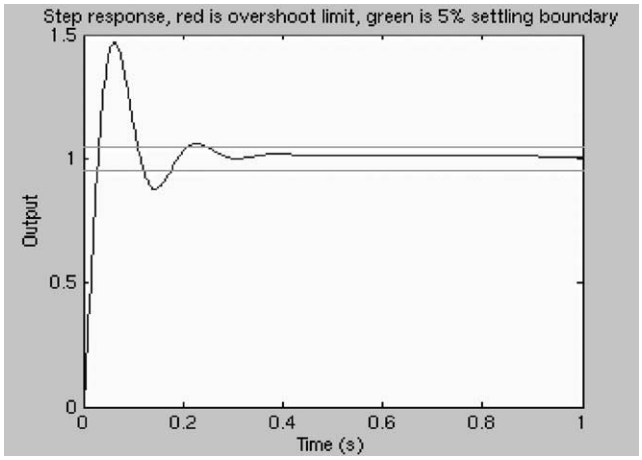

Fig. 11. Step response of controlled pendulum system.

Again, the evaluation was performed on a PC running Linux with an Athlon 700 MHz processor using the approach described in Section 3.3. The total evaluation time was approximately 12 h. Two graphs (Figs. 12 and 13) were obtained from the framework. In both cases, the PID loop gains were not re-tuned for each period because this gave wider variations in behaviour than for a stable system. Retuning every time would have meant comparison between the results at different periods would have been more difficult. Fig. 12 shows the results for constant separation (at 0.5 ms) between sampling and actuation. Fig. 13 shows the results for varying the separation (chosen using a random distribution) between sampling and actuation. Again, in the figures a dot indicates a combination of deadline, period and gains that are valid, and a cross indicates where the period, deadline and gains are not a valid combination.

The effects shown for the Figures generated in this case study are similar to those for the previous one in Section 4.1. However, a key difference is the system can tolerate a higher

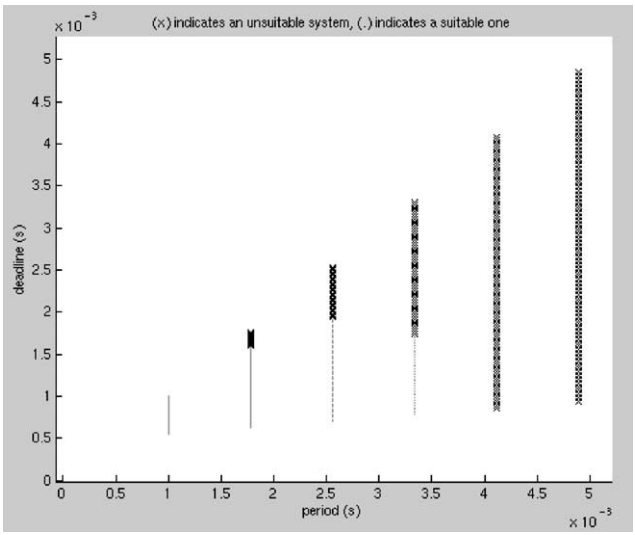

Fig. 12. Deadline versus period with variable sampling jitter.

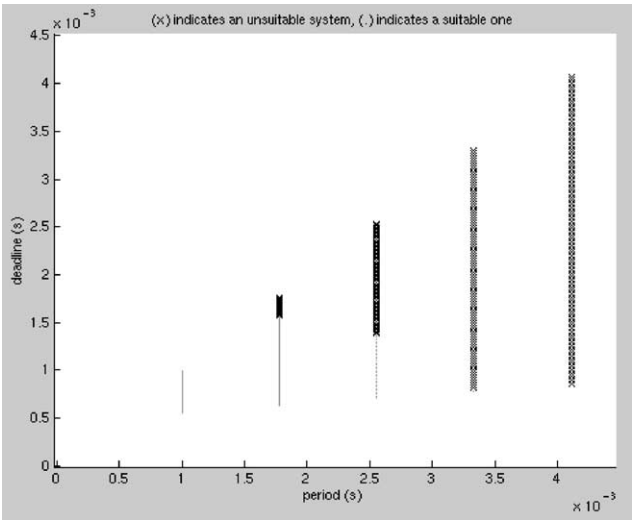

Fig. 13. Deadline versus period with variable separation.

ratio of deadline to period. The reason for this is the behavioural dynamics of the system under control.

4.3. Use of the results

The paper has presented a means by which timing requirements can be determined. The results obtained can be used in many ways. For instance on some projects, a decision may be made not to operate the system at the limits of its ability. For example, if at period T the maximum deadline that has been shown to meet the criteria is D, then a deadline of $0.8D$ could be chosen in-order to give a 'safety' margin. Another way of ensuring the system is not operating at its limits is to input objectives into the framework objectives beyond that actually needed. This would mean the valid solutions that emerge would be on the safe side.

Based on the assumption that a shorter period means we can have a longer deadline and vice versa, the results can be used to help make the system schedulable or scaleable. For instance dependent on the timing requirements associated with the rest of the system, then a larger period may be beneficial in helping reduce processor utilisation. On the other hand for systems with many tasks having short deadlines (making scheduling difficult unless the tasks are phased), it may be better to have a shorter period so that the deadline is longer.

Future work could look at applying genetic algorithms across a number of control loop(s) at the same time as well as other tasks in-order to search for the optimum balance of timing requirements for scheduling and ability to scale the system.

5. Summary and future work

In this paper, a framework for deriving and evaluating the timing requirements for control systems has been presented. For a given control system, the framework automatically evaluates the effect of jitter and period on control properties such as responsiveness and overshoots in-order to derive the set of requirements that meet our criteria, including stability. In other words, it addresses in an integrated way the issue of evaluating the effectiveness of the control system, and the possible realisation of the controller on a computer system. The use of the approach has been illustrated for two very different examples: a car cruise control system that is inherently stable and an inverted pendulum that is unstable without external control.

The work has shown that for a particular system, the control criteria to be satisfied can be met, according to the chosen fitness function, with a wide range of timing requirements. The ability to select the requirements from a variable set can be used to optimise the timing behaviour of the overall system to improve schedulability and scalability.

In general, the advantage of our approach is that it enables control and scheduling issues to be addressed in an integrated way, producing valid and appropriate requirements. If it fulfils its promise then there should be less problems of 'mismatch' between the control systems designers and the software engineers who have to implement their requirements. More specifically, the use of the approach make it less likely that control problems, e.g. instability, are discovered in system testing because the software engineers were working to inappropriate or incomplete requirements. The tools we have developed provide a 'proof of concept'.

Future work could include: determining the optimum balance of timing requirements across multiple control loops and allowing for other resource contentions, further case studies, allowing for important effects such as disturbances (e.g. noise) and non-linear effects, and expanding the framework for so called robust controllers such as $H \infty$ [6]. A robust controller is where the effects of time varying behaviour (e.g. jitter on sampling and actuation) and missed samples are accounted for. The disadvantages of robust controllers are: the extra processing needed due to their increased mathematical complexity, the difficulty in assuring their behaviour in all situations and their use in the real-time systems domain is not normal practice. Supporting robust controllers in the framework would allow comparisons to be performed between the relative effectiveness of PID loops versus robust controllers, and thus give designers a more effective tool for selecting control algorithms and establishing the requirements for their software implementation.

References

[1] C. Liu, J. Layland, Scheduling algorithms for multiprogramming in a hard real-time environment, Journal of the ACM 20 (1) (1973) 40–61.

[2] G. Butazzo, Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications, Kluwer, Dordrecht, 1997.
[3] S. Hutchesson, N. Hayes, Technology transfer and certification issues in safety critical real-time systems, Digest of the IEE Colloquium on Real-time Systems 98/306 (1998).
[4] http://www.mathworks.com
[5] K. Årzé, A. Cervin, J. Eker, L. Sha, An introduction to control and real-time scheduling co-design, Proceedings of the 39th Conference on Decision and Control (2000).
[6] R. Harbor, C. Phillips, Feedback Control Systems, Fourth ed., Prentice Hall, Englewood Cliffs, NJ, 2000.
[7] I. Bate, Scheduling and Timing Analysis for Safety-Critical Systems, Department of Computer Science, University of York, YCST-1998-04, 1998.
[8] M. Torngren, Fundamentals of implementing real-time control applications in distributed computer systems, Real-Time Systems 14 (3) (1997) 219–250.
[9] R. Gerber, S. Hong, M. Saksena, Guaranteeing real-time requirements with resource-based calibration of periodic processes, IEEE Transactions on Software Engineering 21 (7) (1995) 579–592.
[10] M. Nicholson, Selecting a Topology for Safety-Critical Real-Time Control Systems, DPhil Thesis YCST-98-08, Department of Computer Science, University of York, UK, 1998.
[11] http://www.engin.umich.edu/group/ctm/examples/cruise/ccPID.html
[12] http://www.engin.umich.edu/group/ctm/examples/pend/invPID.html
[13] I. Bate, N. Audsley, Architecture trade-off analysis and the influence on component design, Proceedings of Workshop on Component-Based Software Engineering: Composing Systems from Components (2002).
[14] C. Norström, M. Gustafsson, K. Sandström, J. Mäki-Turja, N.-E. Bånkestad, Experiences from introducing state-of-the-art real-time techniques in the automotive industry, Proceedings of the Eighth IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (2001).
[15] I. Bate, T. Kelly, Architectural considerations in the certification of modular systems, Proceedings of 22nd International Conference on Computer safety, Reliability and Security (SAFECOMP 2002) (2002).

Iain Bate has been working as a Research Associate since 1994 and is now a Senior Research Fellow. Prior to this, he worked at British Aerospace for four years after graduating from the University of York with a Masters of Engineering. From 1994 to 1998, Iain worked within the Rolls-Royce University Technology Centre on the industrialisation of fixed priority scheduling. As part of this Iain completed a DPhil titled 'Scheduling and Timing Analysis of Safety Critical Hard Real-Time Systems' and also identified the need for a framework for determining timing requirements of control systems. Between 1999 and 2001, Iain has been working in the BAE SYSTEMS Dependable Computing Systems Centre looking at how advanced processors, such as the Power PC, can be safely and effectively used in safety critical systems, and at architecture assessment methods involving trade-off analysis and optimisation techniques.

John McDermid was appointed to a chair in Software Engineering at the University of York in 1987, where he built up a new research group in high integrity systems engineering (HISE). The group studies a broad range of issues in systems, software and safety engineering, and works closely with the UK aerospace industry. Professor McDermid is the Director of the Rolls-Royce funded University Technology Centre (UTC) in Systems and Software Engineering and the BAE SYSTEMS-funded Dependable Computing System Centre (DCSC). The work in these two centres was a significant factor in the award of the 1996 Queen's Anniversary Prize for Higher and Further Education to the University, and the establishment of a Defence and Aerospace Research Partnership in High Integrity Real-Time Systems based at York. Professor McDermid is author or editor of 6 books, and has published about 280 papers. He was elected a Fellow of the Royal Academy of Engineering in 2002.

Peter W. Nightingale is studying for a Masters in Computer Systems and Software Engineering at the University of York. He worked for two summer vacations at the Dependable Computing System Centre at York on projects related to control and scheduling interaction. His research interests are mainly in artificial intelligence, particularly constraint satisfaction problems and technologies for solving them. These have been expressed through student projects.