Implementation of a PID Controller on a DSP

Åström, Karl Johan; Steingrímsson, Hermann

1990

Document Version:
Publisher's PDF, also known as Version of record

CODEN: LUTFD2/(TFRT-7466)/1–34/(1990)

Implementation of a PID Controller on a DSP

Karl Johan Åström
Hermann Steingrímsson

Department of Automatic Control
Lund Institute of Technology
October 1990

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
INTERNAL REPORT

Date of issue
October 1990

Document Number
CODEN: LUTFD2/(TFRT-7466)/1–34/(1990)

Author(s)
Karl Johan Åström
Hermann Steingrímsson

Supervisor

Sponsoring organisation

Title and subtitle
Implementation of a PID Controller on a DSP

Abstract

This report describes implementation of PID controller on a digital signal processor using fix point calculations. Particular emphasis is given on scaling and testing of the algorithm.

Key words
PID control; DSP; Fix point calculation; Scaling; Testing

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
34

Recipient's notes

Security classification

Implementation of a PID Controller on a DSP†

Karl Johan Åström
Department of Automatic Control
Lund Institute of Technology
Lund, Sweden

Hermann Steingrímsson
Graduate School of Business
University of Wisconsin
Madison, Wisconsin, USA

1. Introduction

The PID controller is by far the most commonly used control algorithm. [Deshpande 1981] Although it is of limited complexity it can be used to solve a large number of industrial control problems. The textbook version of the PID controller can be described by the equation

$$u(t) = K_c\left(e(t) + \frac{1}{T_i}\int^t e(s)ds + T_d\frac{de(t)}{dt}\right) \quad (1)$$

where u is the control variable and e is the control error, defined as $e = y_{sp} - y$, where y_{sp} is the set point and y is the process output. The parameters of the controller are: gain K_c, integral time T_i, and derivative time T_d.

The purpose of the integral action is to increase the low-frequency gain and thus reduce steady-state errors. Derivative action adds phase lead, which improves stability and increases system bandwidth.

Implementation of a PID controller using a DSP will be discussed in this paper. A lot of experience has accumulated over many years of use of the algorithm. This has led to significant modification of the algorithm (1). These modifications will be discussed in Section 2, where the discretization issues are also dealt with. The result is a nonlinear digital algorithm that is suitable for implementation on a general purpose digital computer.

The algorithm can be implemented in a straightforward way in a DSP with floating point hardware. Implementation using an ordinary DSP does, however, require special considerations, because all calculations have to be made in integer arithmetic. These issues are discussed in Section 3.

Some special problems related to quantization in AD- and DA-converters are discussed in Section 4. An overview of the DSP code for a PID controller is described in Section 5. The complete code is given in the Appendix. In Section 6 it is described how the code can be tested. The tests given include both linear and nonlinear behavior.

2. Modification and Discretization

The algorithm (1) has several drawbacks. Significant modifications of linear and nonlinear behavior are necessary in order to obtain a practically useful algorithm. See [Åström and Hägglund 1988]. To obtain equations that can be implemented using computer control it is also necessary to replace continuous time operations like derivation and integration by discrete time operations. See [Åström and Wittenmark 1990]. These modifications will be described in this section.

Proportional Term

The proportional term $K_c e(t)$ is implemented simply by replacing the continuous time variables with their sampled equivalences. One additional modification set point weighting [Åström and Hägglund 1988] has been found useful. This means that the proportional term only acts on a fraction b of the command signal. The proportional term then becomes

$$P(t_k) = K_c\big(by_{sp}(t_k) - y(t_k)\big) \quad (2)$$

where $\{t_k\}$ denotes the sampling instants. The parameter b admits independent adjustment of set point and load disturbance responses. It may also be viewed as "zero-placement".

† Part of this work was done when the first author was visiting professor and the second author a graduate student at the University of Texas at Austin.

Integral Term

When a controller operates over a wide range of operating conditions, the control variable may reach actuator limits. The feedback loop is then broken and the system effectively runs open loop. When this happens in a controller with integral action, the error will continue to be integrated and the integral term may become very large. The integrator "winds up". The error must then change sign for a long period of time to "unwind" the integrator and bring the system back to normal. Windup can also cause problems when the controller is implemented on a microprocessor having finite word length. Since the processor can only store numbers limited in magnitude, windup may cause overflow oscillations in the control variable, unless saturation arithmetic is used.

There are several ways to avoid windup. One possibility is to introduce an extra feedback loop by measuring the output from the actuator and forming an error signal as the difference between the controller output v, and the actuator output u. If the output of the actuator is not available, the signal may be computed by using a mathematical model of the actuator. The error signal is fed to the input of the integrator through the gain $1/T_t$, where the constant T_t is called the tracking time constant. The extra feedback will ensure that the integral obtains a value so that the controller output tracks the saturated output. Tracking is accomplished with the time constant T_t. Using this method of avoiding windup the integral term becomes

$$I(t) = \frac{K_c}{T_i}\int e(s)ds + \frac{1}{T_t}\int \big(u(s) - v(s)\big)ds \quad (3)$$

To obtain an algorithm that can be implemented on a computer, the integral term $I(t)$ is differentiated

$$\frac{dI(t)}{dt} = \frac{K_c}{T_i}e(t) + \frac{1}{T_t}e_s(t)$$

where $e_s(t) = u(t) - v(t)$. Approximating the derivative by a forward difference gives

$$\frac{I(t_{k+1}) - I(t_k)}{h} = \frac{K_c}{T_i}e(t_k) + \frac{1}{T_t}e_s(t_k)$$

where h is the sampling period. Finally, by rearranging terms, we get the following equation to compute the integral term

$$I(t_{k+1}) = I(t_k) + \frac{K_c h}{T_i}e(t_k) + \frac{h}{T_t}e_s(t_k) \quad (4)$$

Derivative Term

A pure derivative should not be implemented, because the controller gain becomes very large at high frequency. This leads to amplification of high-frequency noise. The derivative term is therefore approximated by

$$sT_d \approx \frac{sT_d}{1 + sT_d/N} \quad (5)$$

Notice that the approximation is good for signals whose frequency contents are significantly below N/T_d. Also notice that the approximating transfer function has a maximum gain of N. Parameter N is therefore called maximum derivative gain. In analog controllers N is given a fixed value, typically in the range of 5–20.

It is also advantageous not to let the derivative act on the set point signal. The set point is constant for most of the time and its derivative is therefore zero. A step change in the set point may, however, cause an undesirable jump in the control variable if the derivative acts on the set point. With these modifications the derivative term can be written as

$$D + \frac{T_d}{N}\frac{dD}{dt} = -K_c T_d \frac{dy}{dt} \quad (6)$$

There are several methods to approximate the derivative. Common methods are the forward difference approximation, the backward difference approximation, Tustin's approximation and ramp equivalence. See [Åström and Wittenmark 1990]. These approximations all have the same form

$$D(t_k) = aD(t_{k-1}) - b\big(y(t_k) - y(t_{k-1})\big) \quad (7)$$

and are stable only if $|a| < 1$. The forward difference approximation is stable if $T_d > Nh/2$. It thus becomes unstable for small values of T_d. Tustin's approximation has the disadvantages that a goes to 1 as T_d goes to zero. This gives a ringing response for small T_d. The ramp equivalence approximation gives exact outputs at the sampling instants if the signal is continuous and piece wise linear between the sampling instants, but it requires computations of an exponential. The backward difference approximation gives good results for all values of T_d, The parameter a goes to zero as T_d goes to zero. Here the backward difference approximation is chosen.

The following is obtained when Equation (6) is approximated by a backward difference:

$$D(t_k) + \frac{T_d}{N} \cdot \frac{D(t_k) - D(t_{k-1})}{h} = -K_c T_d \frac{y(t_k) - y(t_{k-1})}{h}$$

Rearranging terms, gives (7) with

$$a = \frac{T_d}{T_d + Nh} \quad \text{and} \quad b = \frac{K_c T_d N}{T_d + Nh}$$

which is the formula that will be used to compute the derivative term.

The PID Algorithm

Summarizing we find that a practical version of the PID algorithm can be described by the following equations:

$$\begin{aligned} P(t_k) &= K_c(by_{sp} - y(t_k)) \\ D(t_k) &= a_d D(t_{k-1}) + b_d(y(t_{k-1}) - y(t_k)) \\ v(t_k) &= P(t_k) + I(t_k) + D(t_k) \\ u(t_k) &= f(v(t_k)) \\ I(t_{k+1}) &= I(t_k) + b_i(y_{sp} - y(t_k)) \\ &\quad + b_t(u(t_k) - v(t_k)) \end{aligned} \tag{8}$$

This algorithm has anti-windup reset, limitation of derivative gain (N) and set point weighting (b).

The function f describes the nonlinear characteristic of the actuator. For a linear actuator with saturation at u_{min} and u_{max} we have

$$f(v(t_k)) = \begin{cases} u_{max} & \text{if } v(t_k) > u_{max} \\ u_{min} & \text{if } v(t_k) < u_{min} \\ v(t_k) & \text{otherwise} \end{cases} \tag{9}$$

For actuators with other limitations the function f should be modified. The parameters a_d, b_d, b_i and b_t are related to the primary parameters K_c, T_i, T_d, T_t and N at the PID controller as follows:

$$\begin{aligned} a_d &= \frac{T_d}{T_d + Nh} \\ b_d &= \frac{K_c N T_d}{T_d + Nh} \\ b_i &= K_c h / T_i \\ b_t &= h / T_t \end{aligned} \tag{10}$$

Since Equations (10) have to be updated only when the controller parameters are changed, the code should be organized so that parameters a_d, b_d, b_i and b_t are computed initially and when the PID parameters are changed. This will reduce the computational load during the execution of the PID algorithm. The structure of the PID algorithm given by Equation (8) is shown in Figure 1. Notice that the algorithm is in parallel form.

The PI algorithm

In many cases the derivative action is not necessary. The algorithm then reduces to

$$\begin{aligned} P(t_k) &= K_c(by_{sp} - y(t_k)) \\ v(t_k) &= P(t_k) + I(t_k) \\ u(t_k) &= f(v(t_k)) \\ I(t_{k+1}) &= I(t_k) + b_i(y_{sp} - y(t_k)) \\ &\quad + b_t(u(t_k) - v(t_k)) \end{aligned} \tag{11}$$

which is a PI controller with anti-windup reset and set point weighting (b).

The function f is the same as in Equation (9) and the parameters b_i and b_t are related to the parameters K_c, T_i and T_t as follows:

$$\begin{aligned} b_i &= K_c h / T_i \\ b_t &= h / T_t \end{aligned} \tag{12}$$

which is the same as Equation (10). The reason for considering this special case is that PI controllers are in fact more common than controllers with derivative action.

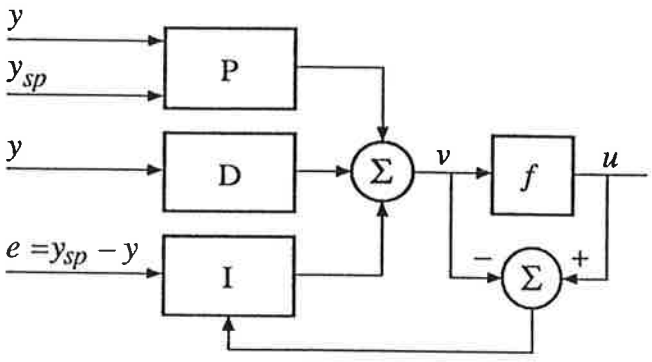


Figure 1. Structure of the PID controller with anti-windup.

Table 1. Number of arithmetic operations for PI and PID control.

	PI		PID	
	M	A	M	A
P	2	1	2	1
D	0	0	2	2
v	0	1	0	2
f	X	X	X	X
I	2	4	2	4
Tot	4	6	6	9

Operations Count

It is a common practice to estimate computation times by a simple operation count. This can be strongly misleading when using fixed point calculation, because much of the computation time may be spent on overflow handling and scaling. Table 1 shows the minimum number of multiplications and additions required for the PID and PI algorithms. The PID algorithm requires 15 arithmetic operations, while the PI algorithm requires 10 operations.

3. Implementation Issues

Implementation of a PID-controller using a DSP with fixed point will now be discussed. General practice on implementing algorithms for DSP are given in [Texas Instruments 1986], [Texas Instruments 1989a], [Texas Instruments 1989b], [Texas Instruments 1990a] and [Texas Instruments 1990b].

To perform fix-point calculations it is necessary to know orders of magnitude of all variables. Simulations were performed to get this information. In the simulations the process model

$$G(s) = \frac{1}{(s+1)^4}$$

was used. Figure 2 shows the step response of the system with parameters $K_c = 0.6$, $T_d = 0.5$, $T_i = 2.2$, $T_t = 0.5$, $N = 8$, and a sampling period of 0.1 s. At the time $t = 0.3$ s a load disturbance of 0.3 V is introduced.

Two C-programs were written to test the effects of scaling and roundoff. One program implements the PID controller in double precision arithmetics with no attempt to simulate the effect of finite word length. The other program simulates

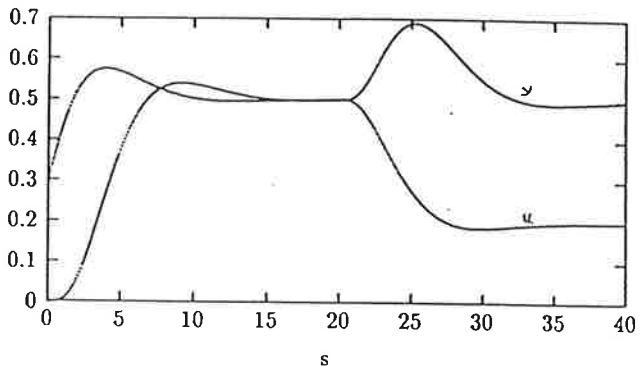


Figure 2. Step response of the system.

the Texas Instruments DSP by using a 32-bit accumulator and a 16-bit word length. The effect of using different resolution of the A/D- and D/A-converters can also be simulated.

Selection of Sampling Period

There are several rules of thumb for choosing the sampling period for digital controllers. For a PI-controller the sampling period is related to the integration time. A rule of thumb [Åström and Wittenmark 1990] is

$$\frac{h}{T_i} \approx 0.1 - 0.3$$

A PID controller requires a much shorter sampling period. The sampling period should be short enough so that the pole $s = -N/T_d$, introduced to limit the high frequency gain of the derivative, can be approximated appropriately. This leads to the following rule of thumb:

$$\frac{hN}{T_d} \approx 0.2 - 0.6$$

See [Åström and Wittenmark 1990].

Integral Offset

Roundoff may give an offset when the integral term is implemented on a computer with a short word length. This can be understood as follows. Consider the equation for the integral term in Equation (8). The correction term $b_c e(t_k) = K_c h/T_i \cdot e(t_k)$ is usually small in comparison to $I(t_k)$ and may therefore be rounded off. With fractional arithmetic, the largest magnitude of the correction term is $K_c h/T_i$. To avoid roundoff, it is therefore necessary to have a word length of at least

$$\text{number of bits} = -\frac{\log(K_c h/T_i)}{\log(2)}$$

More bits are of course required to obtain meaningful values. For example, with $h = 0.02$ s, $T_i = 10$ s and $K_c = 0.1$ the number of bits required to obtain less than 5% error in the integral requires a word length of at least

$$\text{number of bits} = -\frac{\log(0.0002 \cdot 0.05)}{\log(2)} \approx 17$$

Longer sampling periods for computing the integral may be used to avoid the offset. This can be done simply by adding the error over each sampling period and updating the integral term in regular intervals. Another way to avoid offset due to roundoff is to store the integral with higher precision. In most DSPs (like the TMS320xx) values can be stored in double precision, with little overhead.

Scaling

The PID controller given by Equations (8) is already in parallel form, with the modules of zero and first order. Figure 1 illustrates the realization of the controller. Because of the parallel form, the P, I and D terms can be scaled and computed separately and then unified to form v.

Coefficient Scaling

Because of the wide number range of the parameters, some restrictions must be imposed on the magnitude of coefficients. It follows from Equation (10) that b_d is the largest parameter. A limit should therefore be set on the gain K_c, and the high-frequency derivative gain N. If K_c and N are limited to 16, we have $b_d < K_c N = 256$ and $K_c \leq 16$. These parameters must therefore be divided by 256 and 16 respectively before they are stored. To restore the magnitude of the signal, the derivative term must be shifted left by 8 bits and the proportional term shifted left by 4 bits.

The other parameters, a_d, b_i and b_t are within the number range, but because b_i and b_t may become very small, it is advantageous to also set a lower limit on h/T_i and h/T_t.

Signal Scaling and Saturation Arithmetic

It must be insured that overflow does not occur when computing the states of the controller. With the structure of the PID controller shown in Figure 1 the states are $D(t_k)$ and $I(t_{k+1})$. Care must also be taken so that overflow does not occur when the P, I and D terms are added to obtain v.

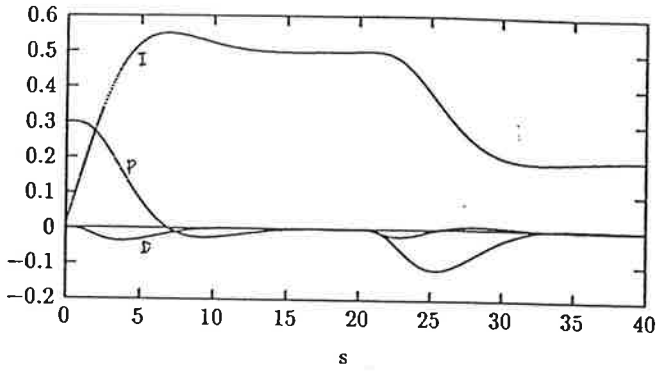


Figure 3. The terms of the PID controller.

The proportional term will always be within the number range, since the multiplication of a fraction with a fraction gives a fraction. Overflow can occur if K_c is larger than 1 when the magnitude of the signal is restored. It is therefore necessary to use saturation arithmetic when computing the proportional term.

One additional advantage of using the anti-windup reset when computing the integral term is that the integral is within the number range. Saturation arithmetic is therefore not necessary. Integration can result in overflow if anti-windup is not used or if T_t is chosen poorly. Saturation arithmetic should therefore be used before the integral is stored.

Since the derivative depends only on the process output, it is difficult to use analytic scaling methods effectively. It is easy to predict the worst possible input, but for most processes that would be too pessimistic. A good engineering approach is therefore to simulate the closed loop system and store the output of the derivative for a few representative examples. The derivative should normally not account for more than 20% of the control signal. Since b_d can take large values, saturation arithmetic should be used before storing the derivative. A number of simulations were made in order to obtain typical orders of magnitude of the proportional, integral and derivative term. It turns out, that under normal operation conditions, the variables are within the number range. Since we are allowing a gain larger than one, it is very likely that an overflow will occur under some operation condition, for example during start-up. Saturation arithmetic is therefore used on both states and on the control signal v. Figure 3 shows Simnon plots of the P, I and D terms for step response and load disturbances, for the process and the controller previously used.

Gain, Input and Output Scaling

To implement a high gain ($K_c > 1$) one can either include the gain in the digital algorithm or move the gain "outside" of the DSP by using a linear amplifier. The advantage of the latter approach is that the control algorithm can be scaled to eliminate the danger of overflow and therefore avoiding the large overhead associated with saturation arithmetic. This gives a shorter code and a faster controller. But there is also a disadvantage. Under normal steady-state operation the error is small and any changes in the control signal will be a relatively small part of the whole dynamic range. A change in the control signal of one quantization step will be amplified, resulting in a large jump. It may also give rise to limit cycles. When a high gain is incorporated in the DSP code, saturation arithmetic must be used on internal calculations.

4. Quantization Effects

Issues related to the interfacing of the DSP to the plant will now be considered. The key questions are related to quantization of A/D- and D/A-converters

Quantization of the Set-Point Value

When implementing the controller the set point should be quantized in the same way as the controller input. That is, the set-point value should either be read through the same, or a similar, A/D-converter as is used for the input signal (if A/D-converter is being used) or quantized internally by using the same resolution as of the A/D-converter. If this is not done there may be an offset or a limit cycle due to the quantization. Figure 4 shows the result of a simulation, when a 6-bits A/D-converter is used for the input signal but the set-point value of 0.455 V is represented with a 16-bit accuracy. The system goes into a limit cycle with a period of 6.77 seconds and an amplitude of 3.8 mV. The reason for this is that the set-point value of 0.455 V can not be represented by the 6-bits A/D-converter. In steady-state the error between the process output and the set-point value will be either 17.5 mV or –13.8 mV. This error will be summed up by the integrator, resulting in a limit cycle.

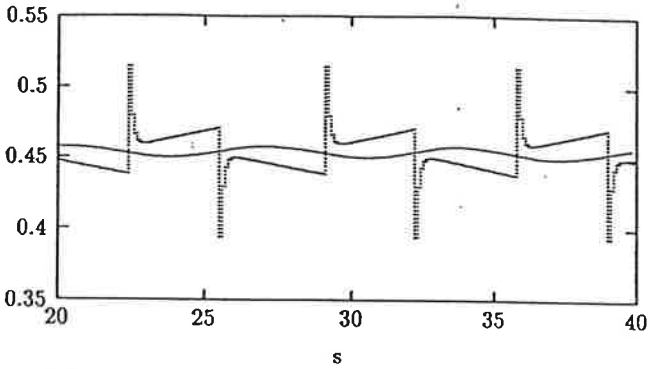


Figure 4. Limit cycles due to high resolution of the set point.

Because the limit cycle is very close to a sinusoid it is reasonable to assume that the period and the amplitude of the limit cycle can be predicted by using describing function analysis. Since the system is in steady-state and the oscillation corresponds to one quantization step of the A/D-converter, we can assume a zero set-point value and model the A/D-converter by a relay nonlinearity centered around zero with the quantization limits +0.00157 and –0.00157. The describing function for this nonlinearity is

$$N(A) = \frac{2q}{\pi a} = \frac{0.0199}{a}$$

where a is the amplitude of the input signal and $q/2$ is half the quantization step. The calculations are simplified if the digital PID-controller is approximated by a continuous-time PI-controller with the transfer function

$$G_c(s) = K + \frac{K}{Ts}$$

where $K = 0.6$ and $T = 2.2$. Possible limit cycle is given by the equation

$$1 + Y_q(A)L(j\omega) = 0$$

Which is equivalent to

$$L(j\omega) = \frac{-1}{N(a)} \qquad (13)$$

where L is the loop transfer function of the controller and the process, in cascade, i.e.

$$L(s) = \frac{K + TKs}{Ts(s+1)^4} \qquad (14)$$

Since the describing is real-valued, one simply has to find the intersection of $L(j\omega)$ with the negative real axis. When $j\omega$ is substituted for s in Equation (14) we get, after separating the real and the imaginary part

$$L(j\omega) = \frac{K\big(A(\omega) + iB(\omega)\big)}{T(4\omega^4 - 4\omega^2)^2 + (\omega^5 - 6\omega^3 + \omega)^2} \quad (15)$$

where $A = T(\omega^6 - 6\omega^4 + \omega^2) + 4\omega^4 - 4\omega^2$ and $B = T(4\omega^5 - 4\omega^3) - \omega^5 + 6\omega^3 - \omega)$. The problem is therefore reduced to finding the frequency where the imaginary part is zero, i.e.

$$7.8\omega^4 - 2.8\omega^2 - 1 = 0 \quad (16)$$

The equation has one positive real root $\omega = 0.7616$, which corresponds to a limit-cycle period of 8.25 s. This is longer than the period $T = 6.77$ s, obtained in the simulation. The amplitude of the limit cycle is then determined by solving Equation (13) for $\omega = 0.7616$, which gives $a = 5.6$ mV. The value $a = 3.8$ mV was obtained in the simulation.

A/D- and D/A-Conversion

If the controller is interfaced to the plant by A/D- and D/A-converters the effect of the resolution of the converters has to be determined. Figure 5 shows the result of one of several simulations where the A/D-converter has a higher resolution than the D/A-converter. A limit cycle was observed in those simulations. Because of the higher resolution of the A/D-converter, the controller produces control signals which are not representable by the D/A-converter. This results in an oscillation over one quantization step of the D/A-converter. This phenomenon can also be predicted by using describing function analysis, where we assume a zero set-point

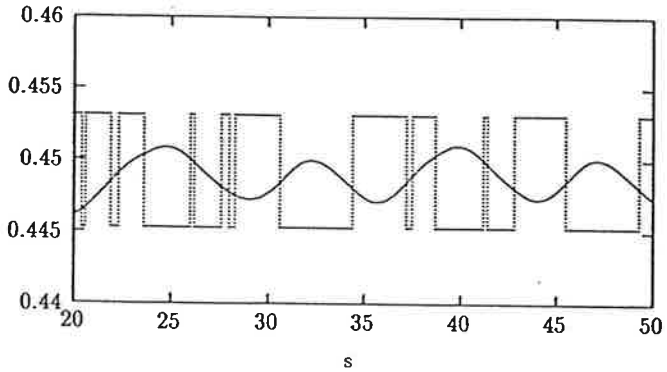


Figure 5. Response with a 10-bit A/D and a 8-bit D/A.

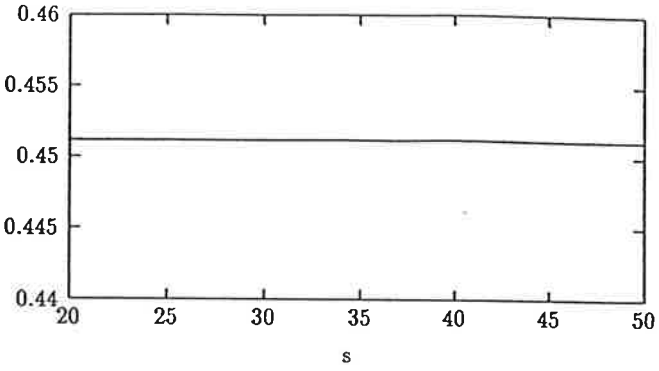


Figure 6. Response with a 8-bit A/D and a 10-bit D/A.

value and the D/A-converter is approximated by a relay. The problem can be avoided by replacing the function f given by Equation (9) by a function that also models the roundoff in the D/A-converter.

Figure 6 shows a good result when an 8-bit A/D-converter and a 10-bit D/A-converter is used when a step input of 0.45 V is applied. These observations indicate that using a D/A-converter with a lower resolution than the A/D-converter may give rise to a limit cycle. It should be emphasized that there are of course many other factors which may be responsible for limit cycles. There are also many other factors that influence the selection of the resolution of the A/D- and D/A-converters, e.g. the required accuracy of the system.

Simulations also showed that a very low resolution (down to 4-bits) of the converters did not have much effect on the step response of the system. The accuracy of the system is, of course, less with low resolution converters. Figure 7 shows the response of the same system when a load disturbance of 0.3 V is introduced at $t = 20$ s.

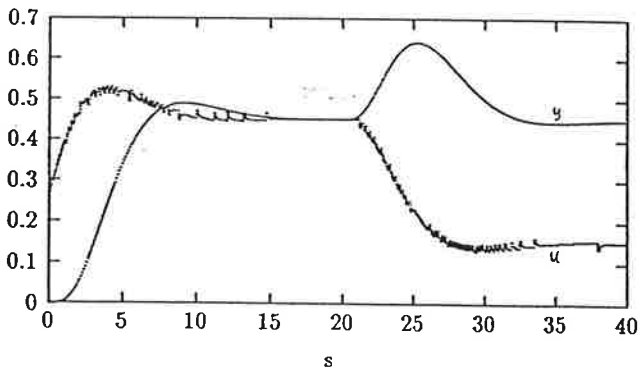


Figure 7. Same as Figure 6 but with a load disturbance.

5. The DSP-Code

To develop and test assembly code of the PID-controller on the Texas Instruments Family of DSPs the Texas Instruments Software Development System (SWDS) was used. This system consists of a PC-board with a TMS320C25 signal processor and PC development environment, which has many features. It is possible to set break-points and single-step through the program. One useful feature is the possibility to specify an input file (or files) to the DSP and to direct the output (or outputs) of the DSP to an output file. This feature makes it easy to test an algorithm, since a predefined input signal can be fed to the controller to test its open loop response.

Programs for PI- and PID-controllers were written for the signal processors TMS32010 and TMS320C25. The complete codes are given in Appendices A, B, C and D. The code for the PID-controller is organized in the following way:

```
INITIALIZE
    load constants from program memory
        to data memory
    clear variables
    load y(n-1) and ysp
    reset external devices (f.ex. analog board)
PID
    wait for input y(n)
    compute derivative (D)
    round off, check for overflow and store D
    compute proportional part (P)
    add D, P and I
    round off, check for overflow and store in v(n)
    compute u(n) from saturation function
    output u(n)
    compute I
    check for overflow and store I
        in double precision
GOTO PID
```

The code for the PI-controller is obtained by deleting the computations of the D-term.

Initialization

After reset the program jumps into the initialization routine. This part disables interrupts, sets overflow mode and loads coefficients from program memory (where they are stored permanently) into data memory. Then the states of the controller are cleared, the set point value (ysp) is read from PA3 and the process output ($y(n-1)$) from PA0. By filling up the y-vector before entering the PID loop a jump due to the derivative is avoided. The program then goes into an infinite loop, to compute the control signal.

PID Calculations

The magnitude of the coefficient b_d of the derivative term is less than 256. To represent it in the DSP it must be scaled by dividing by 256. This can be done by shifts. Before the derivative is stored it is therefore Shifted left by 9 bits (8 bits plus one left shift to account for the extra sign bit which is generated in the multiplication).

The largest proportional gain is 16. The proportional term is therefore divided by 16. It was advantageous also to divide the D and I terms by 16 and restore the signal after the control signal v has been calculated. The same saturation, rounding and shifting can then be applied to both the derivative term and the control signal. Since the derivative must be divided by 16 before it is added to the proportional part, it is advantageous to store a_d divided by 16. A little trick was used to calculate the correct derivative. After $a_d D(t_{k-1})$ has been calculated and stored in the accumulator the term $b_d(y(t_{k-1}) - y(t_k))$ is calculated, and the result is stored in the P register. The value of the P register is then added 16 times to the accumulator to form the correct derivative divided by 16. By doing this in overflow mode, overflow results in saturation of the accumulator. This would not be the case if the value in the accumulator were simply shifted left. With the TMS320C25 adding is easily done using the repeat instruction. After these calculations the derivative is in the accumulator. The proportional term is then added to the accumulator to obtain (P+D)/16. In this way the proportional term does not have to be stored separately.

To obtain the output v, the integral computed previously is divided by 16 by shifting the value right 4 bits. It is then added to P+D in the accumulator. The output then goes through the saturation arithmetic. It is rounded and shifted before it is stored as a 16-bit number. The saturation function f is called to form the final output u.

Since the control signal u depends on the integral from the previous sample, it can be converted to analog form before the integral is updated. This shortens the computational delay between the A/D

Table 2. Cycle count and maximum sampling frequency for PI- and PID-controllers.

DEVICE	PI		PID	
	cycles	KHz	cycles	KHz
TMS32010	94	53	145	34
TMS320C14	94	66	145	43
TMS320C25	89	112	141	70

and D/A-conversions. To avoid integral offset, the integral is computed and stored in double precision. Saturation arithmetic is performed before it is stored, although it is actually not necessary if proper anti-windup is used.

With the chosen method of organizing the calculations the P, D and I terms are added, to form v, with a precision of 27 bits. The terms D and v are then stored with a precision of 16 bits and the integral is calculated and stored with a precision of 31 bits.

Saturation Arithmetic. Before the derivative or the control signal v is stored in memory as a 16-bit value, it must by shifted left by 5 bits, because the signal is divided by 16 in internal calculations and an additional left shift must be performed to account for the extra sign bit generated in the multiplication. The value is rounded and checked for overflow before shifting it. If overflow is detected, the value is replaced by the largest positive or negative number.

Set-Point Value. The set point is read via interrupt. This interrupt is disabled when the control value is computed, but is allowed for a short period, before the next process output is read.

Computation Time

By using the timer on the TMS320C25 it was possible to count the cycles required for one execution of the PID (or PI) loop. To find the number of cycles required for one execution of the TMS32010 (TMS320C14) code, a simple cycle count was done. In all instances it is assumed that the internal memory of the DSPs are used.

Table 2 shows the number of cycles for each controller and the maximum sampling frequency which can be used. From this table we see that the calculation of the derivative consumes a large portion of the total cycles, approximately 50%. The reason for this is that the shifting and saturation arithmetic on the derivative is complicated, because the coefficients of the controller are scaled differently. If the coefficients would all have the same upper limit the same scaling constant could be used and the shifting and saturation arithmetic would be simpler and faster. Table 3 shows how the cycles are divided between different functions of the algorithm. Notice that the division is somewhat arbitrary, because it is not obvious when one operation begins and the other ends. The saturation arithmetic-, rounding- and shifting-function used on the derivative and the output v uses 19 cycles, the saturation arithmetic on the integral uses 10 cycles and the anti-windup function uses 12 cycles.

Table 3. Cycles count for different parts of the PID-controller.

OPERATION	TMS32010 cycles	TMS320C25 cycles
Derivative	9	9
- " - srss	43	45
Proportional	7	7
Integral shifting	12	8
srss on v	23	23
anti-windup	12	12
Integral	15	13
Integral s.a.	10	10
I/O and other	14	14
Total	145	141

srss = saturation, round, shift, store
s.a. = saturation arithmetic

Notice that the code must be modified if K_c and N are to be larger than 16. Also notice that the code can be improved if the parameters of the controller can be limited to smaller ranges. For specific applications, where tighter bounds on parameters and controller states are available, the code can be shortened drastically by removing saturation arithmetic and by simplifying scaling.

It is interesting to note that a crude time estimate, based on the operation count in Table 1, underestimates the computation time by an order of magnitude.

6. Testing

To obtain high quality code it is necessary to develop good testing procedures. The DSP code for the PI and PID controllers were tested by simple laboratory experiments to verify that the controller worked as a proper PID controller. To ensure that the code gives the correct numerical results, the following procedure was introduced. Since a PID

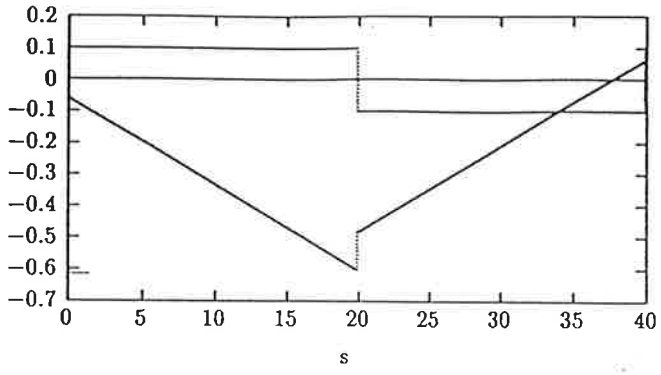


Figure 8. Test of the proportional and integral actions.

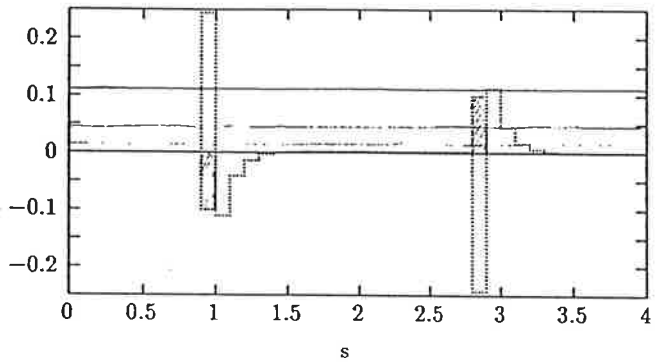


Figure 9. Test of the derivative action.

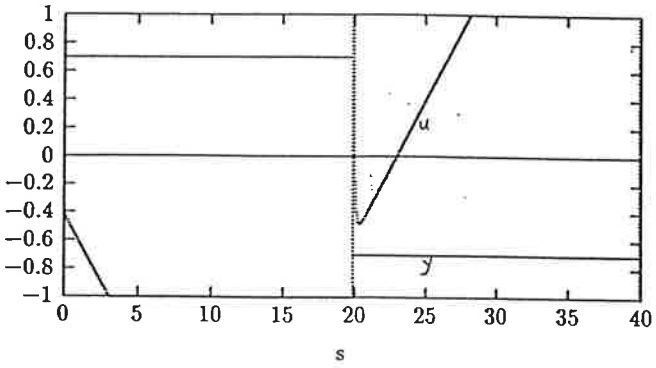


Figure 10. Test of the saturation arithmetic.

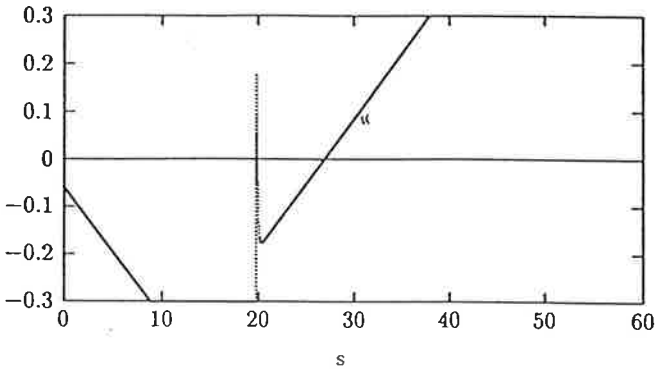


Figure 11. Test of the anti-windup.

controller is a dynamical system, its behavior can be tested by computing its response to given input data with known responses. The test can easily be automated by storing the data in files. This was easily done using the facilities in the Texas Instrument Software Development System. This section describes how the testing was done. The parameters used were $K = 0.6$, $T_d = 0.5$, $T_i = 2.2$, $T_t = 0.5$ and $N = 8$. Parameters a_d, b_d, b_i and b_t were calculated by assuming a sampling period of 0.1 s.

To test proportional and integral action a symmetrical square wave with a period of 40 s and an amplitude of 0.1 V was used as an input sequence. To get a simple case the parameters of the derivative term were set to zero (which is really not necessary, since the derivative dies out very quickly). This sequence can therefore also be used to test a PI controller. Figure 8 shows the input and the resulting output. For a constant input the output of the controller at the time t should be

$$u(t) = \frac{tK_c}{T_i}e + I(0) + K_c e$$

With $I(0) = 0$ the output should be equal to -0.6055 V after 20 seconds. The line $y = -0.6055$ is also drawn in Figure 8 indicating that the proportional and the integral term work properly.

To test the derivative action two impulses, lasting one sampling period, of magnitude -0.1 V and +0.1 V where applied to the input at the time $t = 1$ sec. and $t = 3$ sec. Figure 9 shows the result. The formula for the derivative term is

$$D(t_k) = a_d D(t_{k-1}) + b_d\big(y(t_{k-1}) - y(t_k)\big)$$

If an impulse of magnitude 0.1 V is applied to the derivative we get the sequence: -0.2446, 0.1136, 0.0437, 0.0168, 0.0065,.... The first numbers of this sequence are also plotted on the Figure 9, showing that the derivative action works properly. The small error in the beginning of the second response is due to the integral of the first impulse. This integral is canceled out by the second impulse resulting in a final output equal to zero. To test the saturation arithmetic the amplitude of the input square wave was increased to 0.7 V. Figure 10 shows good results. When the output reaches the limit it is saturated without causing overflow oscillations. Finally, Figure 11 shows the result when the anti-windup reset function is used to limit the output to ± 0.3 V. All versions of the PI- and PID-controller were tested by using these input

sequences. Once a correct set of output files have been obtained one can test modified algorithms simply by comparing the output files, either by plotting the output or by using a file-compare program.

Other testing procedures were also developed using ideas similar to the ones described above.

7. Conclusions

This paper has given algorithms for high quality PI and PID controllers with features like set-point weighting, limitation of derivative gain and anti-windup. It has also been demonstrated how the code can be implemented on a DSP using fix-point calculations. Such an implementation necessarily requires some a priori knowledge of signal and parameter ranges. This means that the code given here only works well in cases that fit the assumptions made.

We have attempted to describe our reasoning in sufficient detail so that the code can be easily adapted to other situations. Some test procedures that we have found useful are also presented. The performance estimates show that PI controller can be executed at 53 kHz on a TMS32010 and at 112 kHz on a TMS320C25.

8. References

Åström, K. J., and T. Hägglund (1988): *Automatic Tuning of PID Controllers*, ISA, Research Triangle Park, NC.

Åström, K. J., and B. Wittenmark (1990): *Computer Controlled Systems – Theory and Design*, Second edition, Prentice-Hall, Englewood Cliffs, NJ.

Deshpande, P. B., and R. H. Ash (1981): *Computer Process Control*, ISA, Research Triangle Park, NC.

Texas Instruments (1986): *Digital Signal Processing Applications with the TMS320 Family – Theory, Algorithms, and Implementations*, Digital Signal Processing, Semiconductor Group.

Texas Instruments (1989a): *TMS320C1x / TMS320C2x – User's Guide*, Digital Signal Processor Products.

Texas Instruments (1989b): *TMS320 Family Development Support – Reference Guide*, Digital Signal Processor Products.

Texas Instruments (1990a): *Digital Signal Processing – Applications with the TMS320 Family*, Application book volume 3, Digital Signal Processor Products.

Texas Instruments (1990b): *TMS320C3x – User's Guide*, Digital Signal Processor Products.

Appendix A: PI-Controller for TMS32010

```
; PI Controller for TMS32010 Version 1.0
; Author: Hermann Steingrimsson
; Date: 3-26-1990
;
;
; RESERVE SPACE IN DATA MEMORY FOR CONSTANTS AND VARIABLES
        .bss    HTE1,1          ;Temporary storages
        .bss    LTE1,1
        .bss    HTE2,1
        .bss    LTE2,1
        .bss    IH,1            ;Integral high
        .bss    IL,1            ;Integral low
        .bss    KC,1            ;Coeff for P
        .bss    KCB,1
        .bss    BI,1            ;Coeff for I
        .bss    BT,1
        .bss    UMAX,1          ;Maximum output
        .bss    UMIN,1          ;Minimum output
        .bss    MODE,1          ;Extra constant
        .bss    CLOCK,1         ;Sampling rate
        .bss    ONE,1           ;One
        .bss    MAXNUM,1        ;Maximum number
        .bss    MINNUM,1        ;Minimum number
DTend   .bss    MINUS,1         ;FFFF
;End of parameters in data memory

        .bss    YN,1            ;y(n)
        .bss    YNM1,1          ;y(n-1)
        .bss    YSP,1           ;y set point
        .bss    UN,1            ;Output
        .bss    VN,1            ;Output before f
        .bss    STA0,1          ;Space to store status register

;Begin program memory

        .sect   "IRUPTS"
        B       START           ;Branch to start of program
        B       ISR             ;Interupt service routine

;Store parameters in program memory

        .data
Ptable  .set    $
        .word   1229,1229,894,6554,9830,-9830,1,1,1,32767,-32768
        .word   -1
Ptend   .set    $-1
SCALE   .set    15
```

```
;Initialize

        .text
START   DINT                    ;Disable interupts
        NOP
        SOVM                    ;Set overflow mode

;Load coeff from prog. mem to data mem. use TBLR (not BLKP) for 1. generation
;devices

        LARK   AR0,DTend        ;AR0 points to end of data block
        LARK   AR1,Ptend-Ptable ;Counter
        LACK   Ptend            ;Beginning address in program memory
LOAD    LARP   AR0              ;Point to AR0
        TBLR   *-,AR1           ;Move, decr. AR0 and point to AR1
        SUB    ONE              ;Subtract one from accumulator
        BANZ   LOAD             ;AR1 not 0 then decr. AR1 and branch
                                ;=> Coeff loaded into data memory

;Initialize variables

        LDPK   IH               ;Point to correct data page
        ZAC                     ;Clear variables
        SACL   IH
        SACL   IL

        OUT   MODE,PA4          ;Init analog board
        OUT   CLOCK,PA5

WAIT1   BIOZ   GET1             ;Load ysp
        B      WAIT1
GET1    IN     YSP,PA3

WAIT2   BIOZ   GET2             ;Load y(n-1)
        B      WAIT2
GET2    IN     YNM1,PA0

;Begin PI

WAIT    BIOZ   GET              ;Wait for input
        B      WAIT
GET     IN     YN,PA0

        ZAC                     ;Clear accumulator

;P-section

        LT     YSP
        MPY    KCB              ;y(n) * KCB
```

```
        LTA   YN                ;acc = y(n)*KCB - ysp*KC
        MPY   KC
        SPAC

        SACH  HTE1              ;Store P temporarily
        SACL  LTE1

        ZALH  IH                ;Shift integral right 4
        ADDS  IL
        SACH  HTE2
        SACL  LTE2
        LAC   LTE2,12
        SACH  LTE2
        LAC   MINUS,12
        XOR   MINUS
        AND   LTE2
        ADD   HTE2,12           ;I in acc rigth shifted 4

        ADDS  LTE1              ;Add P to acc to form P + I
        ADDH  HTE1

        LARK  AR0,VN            ;Point AR0 to VN
        LARP  AR0
        CALL  ROUOF4            ;Round off and overflow check

        CALL  FUNCT             ;Actuator saturation function
        OUT   UN,PA1            ;Output control signal

;I-section

        ZAC
        LT    YSP
        MPY   BI

        LTA   YN
        MPY   BI
        SPAC

        LT    UN
        MPY   BT

        LTA   VN
        MPY   BT
        SPAC

        ADDS  IL                ;Add old I with double precision
        ADDH  IH
```

```
        SACH  IH                ;Store integral
        SACL  IL

        BLZ   INEG              ;Overflow check (10 instr. cycles)
        SUB   MAXNUM,SCALE      ;Subtract maximum pos. number
        BLEZ  OUT4              ;If acc <= 0 then no overflow
        LAC   MAXNUM,SCALE      ;else store maximum number
        SACH  IH
        SACL  IL
        B     OUT5

INEG    SUB   MINNUM,SCALE      ;Subtract maximum neg number
        BGEZ  OUT4              ;If acc >= 0 then no overflow
        LAC   MINNUM,SCALE      ;else store minimum number
        SACH  IH
        SACL  IL
        B     OUT5

OUT4    NOP
        NOP
        NOP
        NOP
        NOP
OUT5    EINT                    ;Enable interupt
        NOP
        NOP
        DINT                    ;Disable interupt
        B     WAIT              ;Loop again

;Rounding and overflow function (11 cycles)

ROUOF4  BLZ   RNEG              ;Check if number negative
        ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MAXNUM,SCALE-4    ;Subtract scaled max pos number
        BLEZ  RNO               ;If acc <= 0 then no overflow
        ZALS  MAXNUM            ;else store max num
        SACL  *
        RET

RNEG    ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MINNUM,SCALE-4    ;Subtract scaled min neg number
        BGEZ  RNO               ;If acc >= 0 then no overflow
        ZALS  MINNUM            ;else store min neg number
        SACL  *
        RET
```

```
RNO     ZALH  HTE1              ;Shift number left 4 before store
        ADDS  LTE1
        SACH  HTE1,4
        SACL  LTE1
        ZALH  LTE1
        SACH  LTE1,4
        ZALH  HTE1
        ADDS  LTE1
        SACH  *,16-SCALE
        RET

;Saturation function (14 instr. cycles)

FUNCT   ZALH  VN                ;Load VN
        SUBH  UMIN
        BLZ   LOWER1            ;Branch if v < umin
        ZALH  VN
        SUBH  UMAX
        BLZ   SAME              ;Branch if v < umax
        B     HIGHER            ;v >= umax

LOWER1  ZALH  UMIN
        SACH  UN                ;u = umin
        NOP                     ;Always same time
        NOP
        NOP
        NOP
        NOP
        NOP
        RET

SAME    ZALH  VN
        SACH  UN                ;u = v
        NOP
        NOP
        RET

HIGHER  ZALH  UMAX
        SACH  UN                ;u = umax
        RET

;Interupt service routine. To read set point value

ISR     SST   STA0              ;Save status
        IN    YSP,PA3           ;Load ysp
        LST   STA0              ;Restore status
        RET                     ;Return
        .endæ
```

Appendix B: PI-Controller for TMS320C25

```
; PI Controller for TMS320C25 Version 1.0
; Author: Hermann Steingrimsson
; Date: 3-26-1990
;
;
; RESERVE SPACE IN DATA MEMORY FOR CONSTANTS AND VARIABLES
        .bss    HTE1,1          ;Temporary storages
        .bss    LTE1,1
        .bss    HTE2,1
        .bss    LTE2,1
        .bss    IH,1            ;Integral high
        .bss    IL,1            ;Integral low
        .bss    KC,1            ;Coeff for P
        .bss    KCB,1
        .bss    BI,1            ;Coeff for I
        .bss    BT,1
        .bss    UMAX,1          ;Maximum output
        .bss    UMIN,1          ;Minimum output
        .bss    MODE,1          ;Extra constant
        .bss    CLOCK,1         ;Sampling rate
        .bss    ONE,1           ;One
        .bss    MAXNUM,1        ;Maximum number
        .bss    MINNUM,1        ;Minimum number
DTend   .bss    MINUS,1         ;FFFF
;End of parameters in data memory

        .bss    YN,1            ;y(n)
        .bss    YNM1,1          ;y(n-1)
        .bss    YSP,1           ;y set point
        .bss    UN,1            ;Output
        .bss    VN,1            ;Output before f
        .bss    STA0,1          ;Space to store status register
        .bss    STA1,1

;Begin program memory

        .sect   "IRUPTS"
        B       START           ;Branch to start of program
        B       ISR             ;Interupt service routine

;Store parameters in program memory

        .data
Ptable  .set    $
        .word   1229,1229,894,6554,9830,-9830,1,1,1,32767,-32768
        .word   -1
Ptend   .set    $-1
SCALE   .set    15
```

```
;Initialize

        .text
START   DINT                    ;Disable interupts
        NOP
        SOVM                    ;Set overflow mode
        SSXM                    ;Set sign-extension mode
        SPM   0                 ;No shifting from P register

;Load coeff from prog. mem to data mem.

        LRLK   AR0,DTend        ;AR0 points to end of data block
        LARK   AR1,Ptend-Ptable ;Counter
        LALK   Ptend            ;Beginning address in program memory
LOAD    LARP  AR0               ;Point to AR0
        TBLR  *-,AR1            ;Move, decr. AR0 and point to AR1
        SUBK  1                 ;Subtract one from accumulator
        BANZ  LOAD              ;AR1 not 0 then decr. AR1 and branch
                                ;=> Coeff loaded into data memory

;Initialize variables

        LDPK  IH                ;Point to correct data page
        ZAC                     ;Clear variables
        SACL  IH
        SACL  IL

        OUT  MODE,PA4           ;Init analog board
        OUT  CLOCK,PA5

;WAIT1  BIOZ  GET1              ;Load ysp
;       B     WAIT1
GET1    IN    YSP,PA3

;Begin PID

;WAIT   BIOZ  GET               ;Wait for input
;       B     WAIT
WAIT    IN    YN,PA0            ;Change WAIT to GET when ; are removed

;P-section

        LT    YSP
        MPY   KCB               ;y(n) * KCB

        LTP   YN                ;acc = y(n)*KCB - ysp*KC
        MPY   KC
        SPAC
```

```
        SACH  HTE1              ;Store P
        SACL  LTE1

        ZALH  IH                ;Shift integral right 4
        ADDS  IL                ;because coeff of P where divided by 16
        SFR
        SFR
        SFR
        SFR

        ADDS  LTE1              ;Add P to acc to form P + I
        ADDH  HTE1

        LRLK  AR0,VN            ;Point AR0 to VN
        LARP  AR0
        CALL  ROUOF4            ;Round off and overflow check

        CALL  FUNCT             ;Actuator saturation function
        OUT   UN,PA1            ;Output control signal

;I-section

        LT    YSP
        MPY   BI

        LTP   YN
        MPY   BI

        LTS   UN
        MPY   BT

        LTA   VN
        MPY   BT
        SPAC

        ADDS  IL                ;Add old I with double precision
        ADDH  IH

        SACH  IH                ;Store integral
        SACL  IL

        BLZ   INEG              ;Overflow check (10 instr. cycles)
        SUB   MAXNUM,SCALE      ;Subtract maximum pos. number
        BLEZ  OUT4              ;If acc <= 0 then no overflow
        LAC   MAXNUM,SCALE      ;else store maximum number
        SACH  IH
        SACL  IL
        B     OUT5
```

```
INEG    SUB   MINNUM,SCALE      ;Subtract maximum neg number
        BGEZ  OUT4              ;If acc >= 0 then no overflow
        LAC   MINNUM,SCALE      ;else store minimum number
        SACH  IH
        SACL  IL
        B     OUT5

OUT4    NOP
        NOP
        NOP
        NOP
        NOP
OUT5    EINT                    ;Enable interupt
        NOP
        NOP
        DINT                    ;Disable interupt
        B     WAIT              ;Loop again

; Rounding, overflow and shifting function (13 cycles)

ROUOF4  BLZ   RNEG              ;Check if number negative
        ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MAXNUM,SCALE-4    ;Subtract scaled max pos number
        BLEZ  RNO               ;If acc <= 0 then no overflow
        ZALS  MAXNUM            ;else store max num
        SACL  *
        NOP
        RET

RNEG    ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MINNUM,SCALE-4    ;Subtract scaled min neg number
        BGEZ  RNO               ;If acc >= 0 then no overflow
        ZALS  MINNUM            ;else store min neg number
        SACL  *
        NOP
        RET

RNO     ZALH  HTE1              ;Shift number left 4+1 before store
        ADDS  LTE1
        SACH  *,5
        RET

;Saturation function (12 instr. cycles)
```

```
FUNCT   ZALH  VN              ;Load VN
        SUBH  UMIN
        BLZ   LOWER1          ;Branch if v < umin
        ZALH  VN
        SUBH  UMAX
        BLZ   SAME            ;Branch if v < umax
        ZALH  UMAX            ;v >= umax
        SACH  UN              ;u = umax
        RET

LOWER1  ZALH  UMIN
        SACH  UN              ;u = umin
        NOP                   ;Always same time
        NOP
        NOP
        NOP
        RET

SAME    ZALH  VN
        SACH  UN              ;u = v
        RET


;Interupt service routine. To read set point value

ISR     SST   STA0            ;Save status
        SST1  STA1
        IN    YSP,PA3         ;Load ysp
        LST   STA0            ;Restore status
        LST1  STA1
        RET                   ;Return
        .end
```

Appendix C: PID-Controller for TMS32010

```
; PID Controller for TMS32010 Version 1.0
; Roundoff Corrected
; Hermann Steingrimsson
; Date: 3-26-1990
;
;
; ad and Kc must be divided by 16 before stored
; bd must be divided by 256 before storage
;
; RESERVE SPACE IN DATA MEMORY FOR CONSTANTS AND VARIABLES
        .bss    HTE1,1          ;Temporary storages
        .bss    LTE1,1
        .bss    HTE2,1
        .bss    LTE2,1
        .bss    IH,1            ;Integral high
        .bss    IL,1            ;Integral low
        .bss    DH,1            ;Derivative high
        .bss    KC,1            ;Coeff for P
        .bss    KCB,1
        .bss    BI,1            ;Coeff for I
        .bss    BT,1
        .bss    BD,1            ;Coeff for D
        .bss    AD,1
        .bss    UMAX,1          ;Maximum output
        .bss    UMIN,1          ;Minimum output
        .bss    MODE,1          ;Extra constant
        .bss    CLOCK,1         ;Sampling rate
        .bss    ONE,1           ;One
        .bss    MAXNUM,1        ;Maximum number
        .bss    MINNUM,1        ;Minimum number
DTend   .bss    MINUS,1         ;FFFF
;End of parameters in data memory

        .bss    YN,1            ;y(n)
        .bss    YNM1,1          ;y(n-1)
        .bss    YSP,1           ;y set point
        .bss    UN,1            ;Output
        .bss    VN,1            ;Output before f
        .bss    STA0,1          ;Space to store status register

;Begin program memory

        .sect   "IRUPTS"
        B       START           ;Branch to start of program
        B       ISR             ;Interupt service routine

;Store parameters in program memory
```

```
        .data
Ptable  .set    $
        .word   1229,1229,894,6554,236,788,9830,-9830,1,1,1,32767,-32768
        .word   -1
Ptend   .set    $-1
SCALE   .set    15

;Initialize

        .text
START   DINT                    ;Disable interupts
        NOP
        SOVM                    ;Set overflow mode

;Load coeff from prog. mem to data mem. use TBLR (not BLKP) for 1. generation
;devices

        LARK    AR0,DTend       ;AR0 points to end of data block
        LARK    AR1,Ptend-Ptable ;Counter
        LACK    Ptend           ;Beginning address in program memory
LOAD    LARP    AR0             ;Point to AR0
        TBLR    *-,AR1          ;Move, decr. AR0 and point to AR1
        SUB     ONE             ;Subtract one from accumulator
        BANZ    LOAD            ;AR1 not 0 then decr. AR1 and branch
                                ;=> Coeff loaded into data memory

;Initialize variables

        LDPK    IH              ;Point to correct data page
        ZAC                     ;Clear variables
        SACL    IH
        SACL    IL
        SACL    DH

        OUT     MODE,PA4        ;Init analog board
        OUT     CLOCK,PA5

WAIT1   BIOZ    GET1            ;Load ysp
        B       WAIT1
GET1    IN      YSP,PA3

WAIT2   BIOZ    GET2            ;Load y(n-1)
        B       WAIT2
GET2    IN      YNM1,PA0

;Begin PID

WAIT    BIOZ    GET             ;Wait for input
        B       WAIT
```

```
GET     IN    YN,PA0          ;Change WAIT to GET when ; are removed

;D-section

        ZALH  YNM1            ;y(n-1) - y(n)
        SUBH  YN
        SACH  HTE1            ;Store difference
        DMOV  YN              ;Copy YN into YNM1

        LT    DH              ;ad*D (ad was divided by 16)
        MPY   AD
        PAC

        LT    HTE1            ;difference * bd
        MPY   BD

        APAC    ;1            ;Since bd was divided by 256, bd*diff is
        APAC    ;2            ;added 16 times to the accumulator to
        APAC    ;3            ;form D divided by 16. By doing this the
        APAC    ;4            ;overflow mode will take care of overflow
        APAC    ;5
        APAC    ;6
        APAC    ;7
        APAC    ;8
        APAC    ;9
        APAC    ;10
        APAC    ;11
        APAC    ;12
        APAC    ;13
        APAC    ;14
        APAC    ;15
        APAC    ;16


        SACH  HTE2            ;Store derivative
        SACL  LTE2
        LARK  AR0,DH          ;Point to DH
        LARP  AR0
        CALL  ROUOF4          ;Check for overfl. shift and store
        ZALH  HTE2            ;Restore the derivative
        ADDS  LTE2

;P-section

        LT    YSP
        MPY   KCB             ;y(n) * KCB

        LTA   YN              ;acc = y(n)*KCB - ysp*KC
        MPY   KC
```

```
        SPAC

        SACH  HTE1              ;Store P + D
        SACL  LTE1

        ZALH  IH                ;Shift integral right 4
        ADDS  IL
        SACH  HTE2
        SACL  LTE2
        LAC   LTE2,12
        SACH  LTE2
        LAC   MINUS,12
        XOR   MINUS
        AND   LTE2
        ADD   HTE2,12           ;I in acc right shifted 4

        ADDS  LTE1              ;Add P + I to acc to form P + I + D
        ADDH  HTE1

        LARK  AR0,VN            ;Point AR0 to VN
        LARP  AR0
        CALL  ROUOF4            ;Round off and overflow check

        CALL  FUNCT             ;Actuator saturation function
        OUT   UN,PA1            ;Output control signal

;I-section

        ZAC
        LT    YSP
        MPY   BI

        LTA   YN
        MPY   BI
        SPAC

        LT    UN
        MPY   BT

        LTA   VN
        MPY   BT
        SPAC

        ADDS  IL                ;Add old I with double precision
        ADDH  IH

        SACH  IH                ;Store integral
        SACL  IL
```

```
        BLZ   INEG              ;Overflow check (10 instr. cycles)
        SUB   MAXNUM,SCALE      ;Subtract maximum pos. number
        BLEZ  OUT4              ;If acc <= 0 then no overflow
        LAC   MAXNUM,SCALE      ;else store maximum number
        SACH  IH
        SACL  IL
        B     OUT5

INEG    SUB   MINNUM,SCALE      ;Subtract maximum neg number
        BGEZ  OUT4              ;If acc >= 0 then no overflow
        LAC   MINNUM,SCALE      ;else store minimum number
        SACH  IH
        SACL  IL
        B     OUT5

OUT4    NOP
        NOP
        NOP
        NOP
        NOP
OUT5    EINT                    ;Enable interupt
        NOP
        NOP
        DINT                    ;Disable interupt
        B     WAIT              ;Loop again

; Rounding, overflow and shifting function (19 cycles)

ROUOF4  BLZ   RNEG              ;Check if number negative
        ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MAXNUM,SCALE-4    ;Subtract scaled max pos number
        BLEZ  RNO               ;If acc <= 0 then no overflow
        ZALS  MAXNUM            ;else store max num
        SACL  *
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        RET

RNEG    ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MINNUM,SCALE-4    ;Subtract scaled min neg number
```

```
        BGEZ  RNO               ;If acc >= 0 then no overflow
        ZALS  MINNUM            ;else store min neg number
        SACL  *
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        RET

RNO     ZALH  HTE1              ;Shift number left 4 before store
        ADDS  LTE1
        SACH  HTE1,4
        SACL  LTE1
        ZALH  LTE1
        SACH  LTE1,4
        ZALH  HTE1
        ADDS  LTE1
        SACH  *,16-SCALE
        RET

;Saturation function (12 instr. cycles)

FUNCT   ZALH  VN                ;Load VN
        SUBH  UMIN
        BLZ   LOWER1            ;Branch if v < umin
        ZALH  VN
        SUBH  UMAX
        BLZ   SAME              ;Branch if v < umax
        ZALH  UMAX              ;v >= umax
        SACH  UN                ;u = umax
        RET

LOWER1  ZALH  UMIN
        SACH  UN                ;u = umin
        NOP                     ;Always same time
        NOP
        NOP
        NOP
        RET

SAME    ZALH  VN
        SACH  UN                ;u = v
        RET


;Interupt service routine. To read set point value
```

```
ISR     SST   STA0            ;Save status
        IN    YSP,PA3         ;Load ysp
        LST   STA0            ;Restore status
        RET                   ;Return
        .endæ
```

Appendix D: PID-Controller for TMS320C25

```
; PID Controller for TMS320C25
; Roundoff Corrected
; Author: Hermann Steingrimsson
; Date: 3-26-1990
;
;
; ad and Kc must be divided by 16 before stored
; bd must be divided by 256 before storage
;
; RESERVE SPACE IN DATA MEMORY FOR CONSTANTS AND VARIABLES
        .bss    HTE1,1          ;Temporary storages
        .bss    LTE1,1
        .bss    HTE2,1
        .bss    LTE2,1
        .bss    IH,1            ;Integral high
        .bss    IL,1            ;Integral low
        .bss    DH,1            ;Derivative high
DTbeg   .bss    KC,1            ;Coeff for P
        .bss    KCB,1
        .bss    BI,1            ;Coeff for I
        .bss    BT,1
        .bss    BD,1            ;Coeff for D
        .bss    AD,1
        .bss    UMAX,1          ;Maximum output
        .bss    UMIN,1          ;Minimum output
        .bss    MODE,1          ;Extra constant
        .bss    CLOCK,1         ;Sampling rate
        .bss    ONE,1           ;One
        .bss    MAXNUM,1        ;Maximum number
        .bss    MINNUM,1        ;Minimum number
DTend   .bss    MINUS,1         ;FFFF
;End of parameters in data memory

        .bss    YN,1            ;y(n)
        .bss    YNM1,1          ;y(n-1)
        .bss    YSP,1           ;y set point
        .bss    UN,1            ;Output
        .bss    VN,1            ;Output before f
        .bss    STA0,1          ;Space to store status register
        .bss    STA1,1

;Begin program memory

        .sect   "IRUPTS"
        B       START           ;Branch to start of program
        B       ISR             ;Interupt service routine

;Store parameters in program memory
```

```
        .data
Ptable  .set    $
        .word   1229,1229,894,6554,236,788,9830,-9830,1,1,1,32767,-32768
        .word   -1
Ptend   .set    $-1
SCALE   .set    15

;Initialize

        .text
START   DINT                    ;Disable interupts
        NOP
        SOVM                    ;Set overflow mode
        SSXM                    ;Set sign-extension mode
        SPM   0                 ;No shifting from P register

;Load coeff from prog. mem to data mem. use BLKP

        LRLK  AR0,DTbeg         ;AR0 points to end of data block
        LARP  AR0
        RPTK  Ptend-Ptable      ;Set up counter
        BLKP  Ptable,*+         ;Move data
                                ;=> Coeff loaded into data memory

;Initialize variables

        LDPK  IH                ;Point to correct data page
        ZAC                     ;Clear variables
        SACL  IH
        SACL  IL
        SACL  DH

        OUT  MODE,PA4           ;Init analog board
        OUT  CLOCK,PA5

WAIT1   BIOZ  GET1              ;Load ysp
        B     WAIT1
GET1    IN    YSP,PA3

WAIT2   BIOZ  GET2              ;Load y(n-1)
        B     WAIT2
GET2    IN    YNM1,PA0

;Begin PID

WAIT    BIOZ  GET               ;Wait for input
        B     WAIT
GET     IN    YN,PA0            ;Change WAIT to GET when ; are removed
```

```
;D-section

        ZALH  YNM1              ;y(n-1) - y(n)
        SUBH  YN
        SACH  HTE1              ;Store difference
        DMOV  YN                ;Copy YN into YNM1

        LT    DH                ;ad*D (ad was divided by 16)
        MPY   AD

        LTP   HTE1              ;difference * bd, and store previous product
        MPY   BD

        RPTK  15                ;Since bd was divided by 256, bd*diff is
        APAC                    ;added 16 times to the accumulator to
                                ;form D divided by 16. By doing this the
                                ;overflow mode will take care of overflow

        SACH  HTE2              ;Store derivative
        SACL  LTE2
        LRLK  AR0,DH            ;Point to DH
        LARP  AR0
        CALL  ROUOF4            ;Check for overfl. shift and store
        ZALH  HTE2              ;Restore the derivative
        ADDS  LTE2

;P-section

        LT    YSP
        MPY   KCB               ;y(n) * KCB

        LTA   YN                ;acc = y(n)*KCB - ysp*KC
        MPY   KC
        SPAC

        SACH  HTE1              ;Store P + D
        SACL  LTE1

;P + D are now divided by 16 => shift integral right 4 bits before adding
;to P + D

        ZALH  IH                ;Shift integral right 4
        ADDS  IL
        SFR
        SFR
        SFR
        SFR
```

```
        ADDS  LTE1              ;Add P + I to acc to form P + I + D
        ADDH  HTE1

        LRLK  AR0,VN            ;Point AR0 to VN
        LARP  AR0
        CALL  ROUOF4            ;Round off and overflow check

        CALL  FUNCT             ;Actuator saturation function
        OUT   UN,PA1            ;Output control signal

;I-section

        LT    YSP
        MPY   BI

        LTP   YN
        MPY   BI

        LTS   UN
        MPY   BT

        LTA   VN
        MPY   BT
        SPAC

        ADDS  IL                ;Add old I with double precision
        ADDH  IH

        SACH  IH                ;Store integral
        SACL  IL

        BLZ   INEG              ;Overflow check (10 instr. cycles)
        SUB   MAXNUM,SCALE      ;Subtract maximum pos. number
        BLEZ  OUT4              ;If acc <= 0 then no overflow
        LAC   MAXNUM,SCALE      ;else store maximum number
        SACH  IH
        SACL  IL
        B     OUT5

INEG    SUB   MINNUM,SCALE      ;Subtract maximum neg number
        BGEZ  OUT4              ;If acc >= 0 then no overflow
        LAC   MINNUM,SCALE      ;else store minimum number
        SACH  IH
        SACL  IL
        B     OUT5

OUT4    NOP
        NOP
        NOP
```

```
        NOP
        NOP
OUT5    EINT                    ;Enable interupt
        NOP
        NOP
        DINT                    ;Disable interupt
        B     WAIT              ;Loop again

; Rounding, overflow and shifting function (19 cycles)

ROUOF4  BLZ   RNEG              ;Check if number negative
        ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MAXNUM,SCALE-4    ;Subtract scaled max pos number
        BLEZ  RNO               ;If acc <= 0 then no overflow
        ZALS  MAXNUM            ;else store max num
        SACL  *
        NOP
        RET

RNEG    ADD   ONE,SCALE-5       ;Round
        SACH  HTE1              ;Store value
        SACL  LTE1
        SUB   MINNUM,SCALE-4    ;Subtract scaled min neg number
        BGEZ  RNO               ;If acc >= 0 then no overflow
        ZALS  MINNUM            ;else store min neg number
        SACL  *
        NOP
        RET

RNO     ZALH  HTE1              ;Shift number left 4 before store
        ADDS  LTE1
        SACH  *,5               ;+1 shift because of sign
        RET

;Saturation function (12 instr. cycles)

FUNCT   ZALH  VN                ;Load VN
        SUBH  UMIN
        BLZ   LOWER1            ;Branch if v < umin
        ZALH  VN
        SUBH  UMAX
        BLZ   SAME              ;Branch if v < umax
        ZALH  UMAX              ;v >= umax
        SACH  UN                ;u = umax
        RET

LOWER1  ZALH  UMIN
```

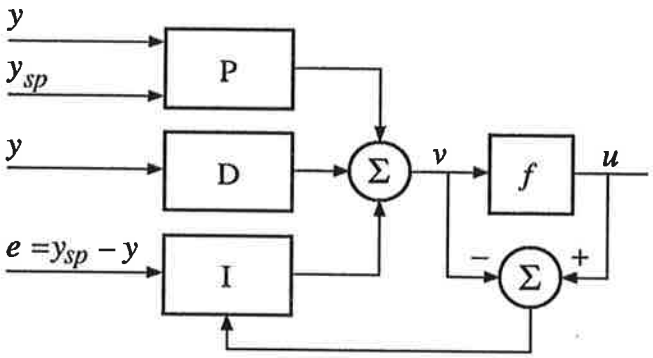
```
        SACH  UN                ;u = umin
        NOP                     ;Always same time
        NOP
        NOP
        NOP
        RET

SAME    ZALH  VN
        SACH  UN                ;u = v
        RET


;Interupt service routine. To read set point value

ISR     SST   STA0              ;Save status
        SST1  STA1
        IN    YSP,PA3           ;Load ysp
        LST   STA0              ;Restore status
        LST1  STA1
        RET                     ;Return
        .endæ
```

Fig. Orginal till
TFRT-7466



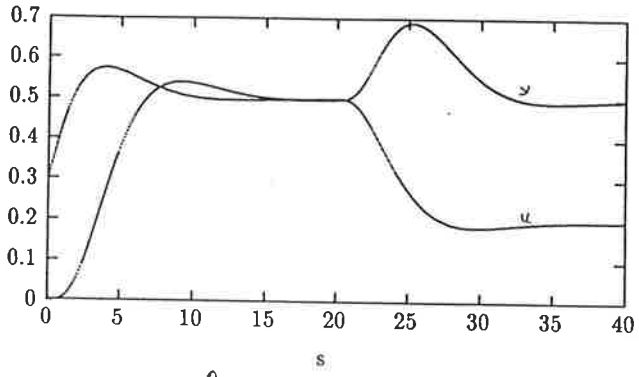


Figure 4.2: Step response of the system

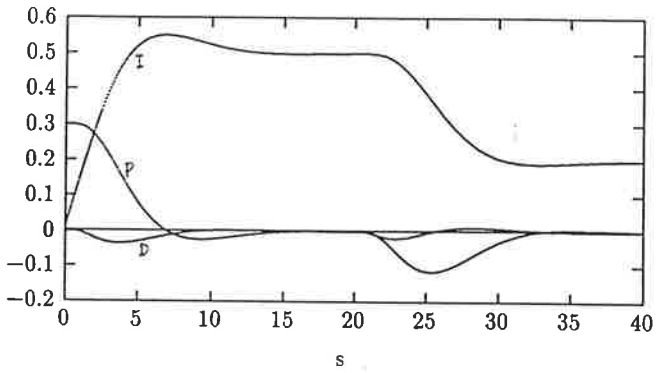


Figure 4.3: The terms of the PID controller

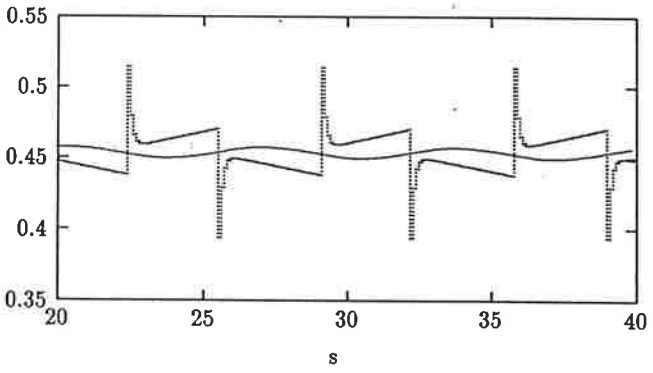


Figure 4.4: Limit cycles due to high resolution of the set-point

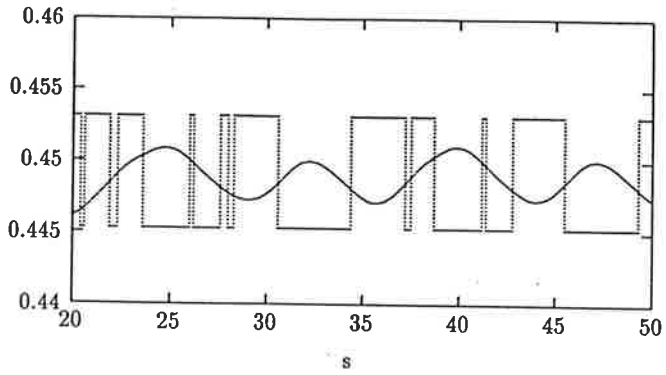


Figure 4.5: Response with a 10-bit A/D and a 8-bit D/A

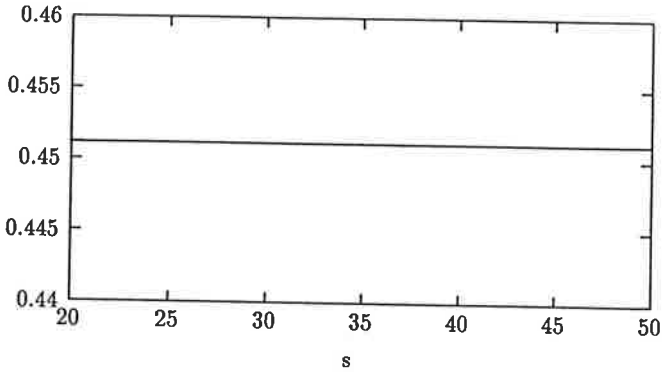


Figure 4.6: Response with a 8-bit A/D and a 10-bit D/A

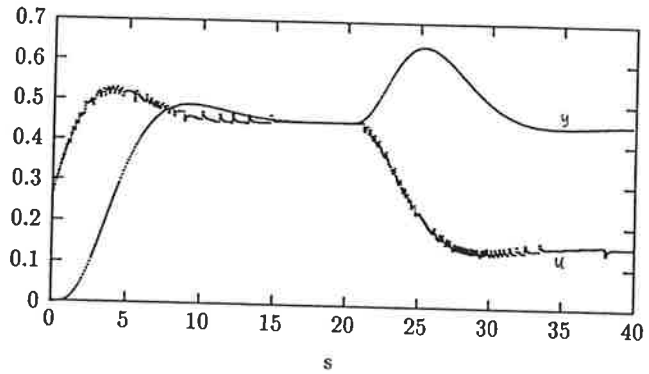


Figure 4.7: Same as 4.6 but with a load disturbance

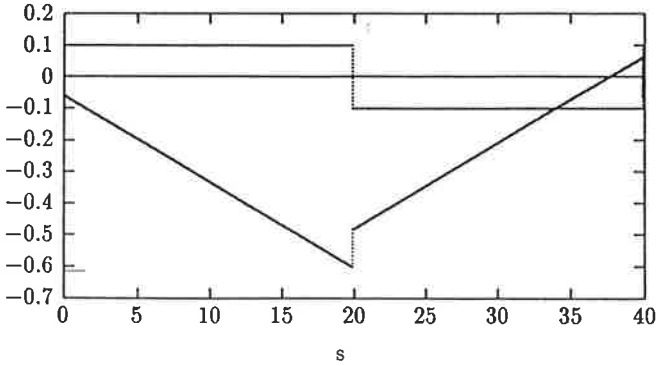


Figure 4.8: Testing proportional and integral action

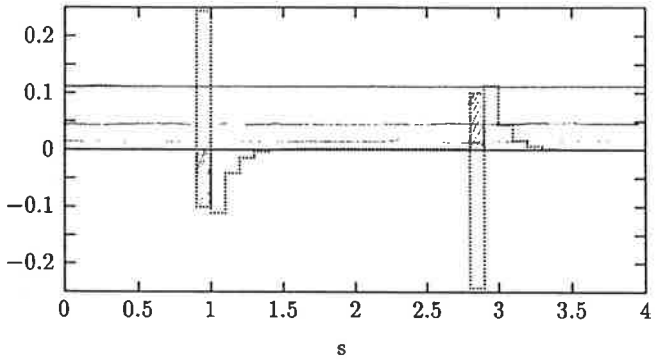


Figure 4.9: Testing derivative action

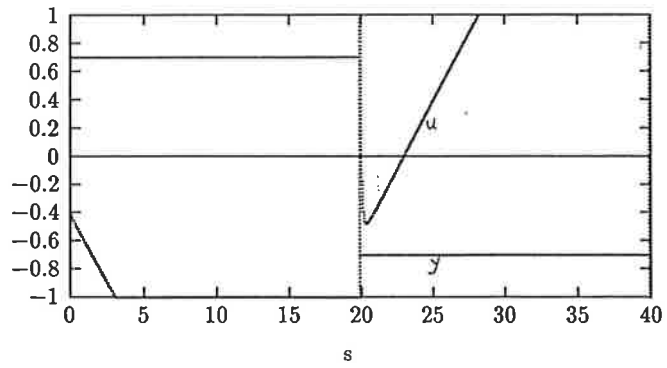


Figure 4.10: Testing saturation arithmetic

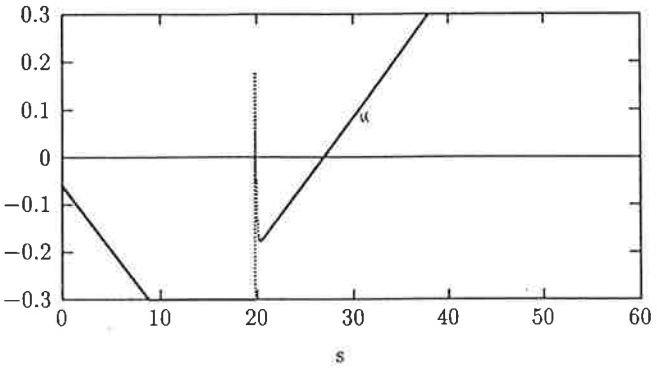


Figure 4.11: Testing anti-windup reset

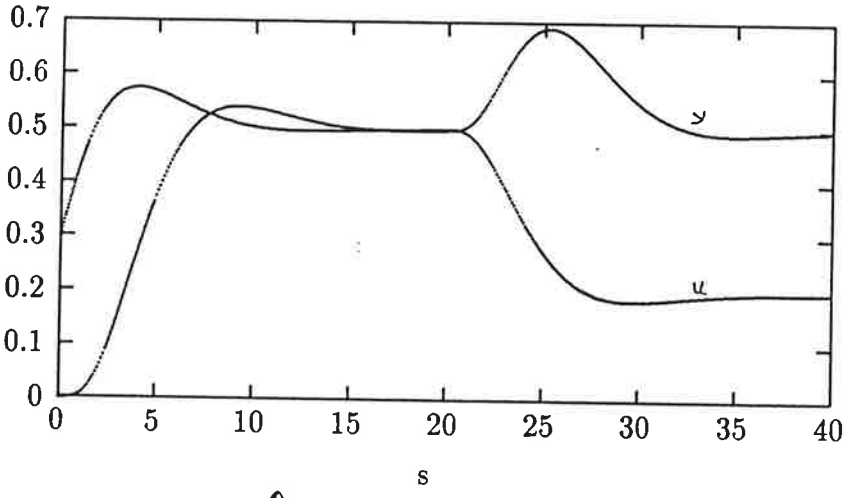


Figure 4.2: Step response of the system

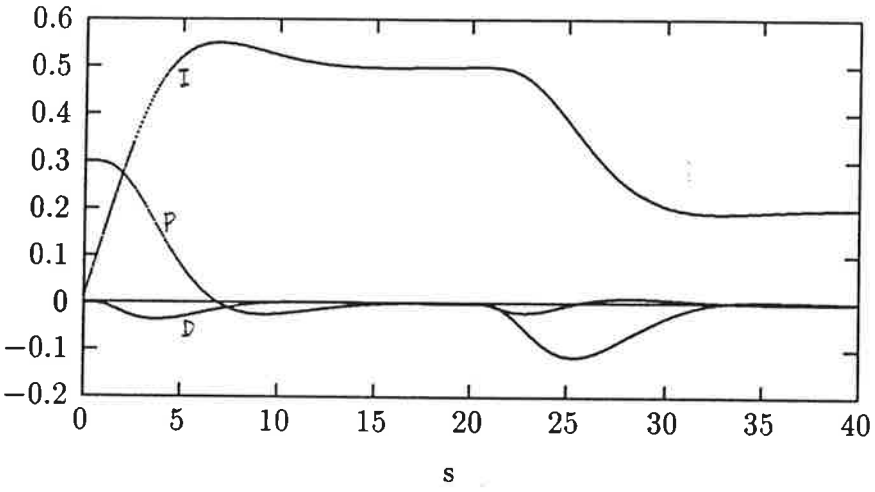


Figure 4.3: The terms of the PID controller

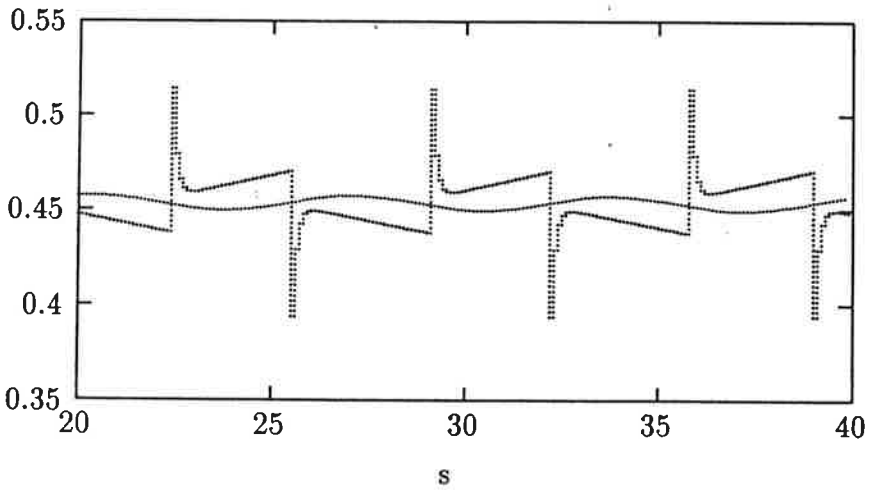


Figure 4.4: Limit cycles due to high resolution of the set-point

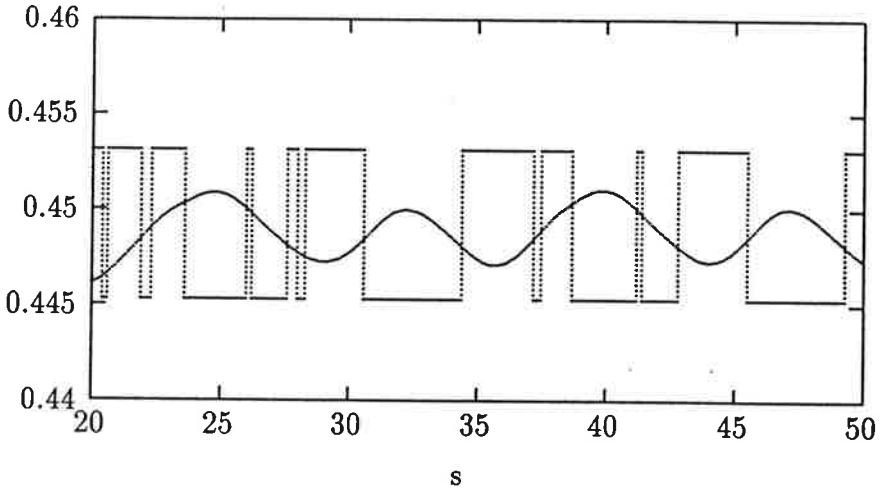


Figure 4.5: Response with a 10-bit A/D and a 8-bit D/A

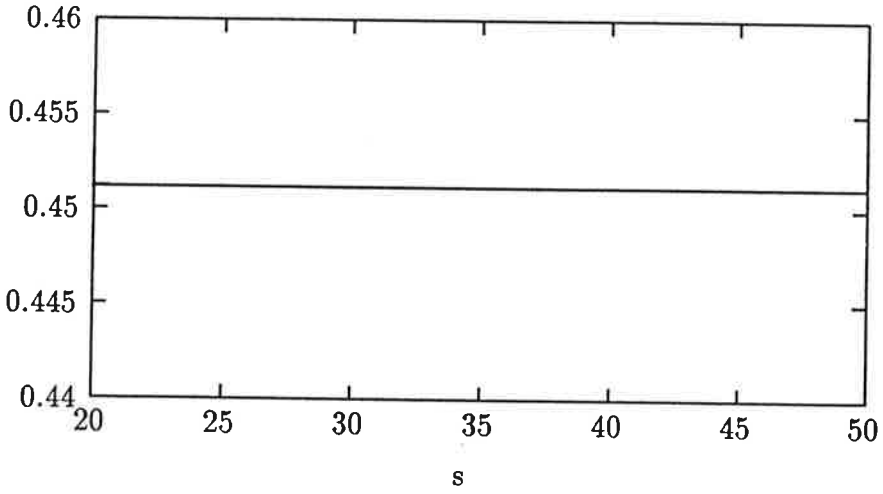


Figure 4.6: Response with a 8-bit A/D and a 10-bit D/A

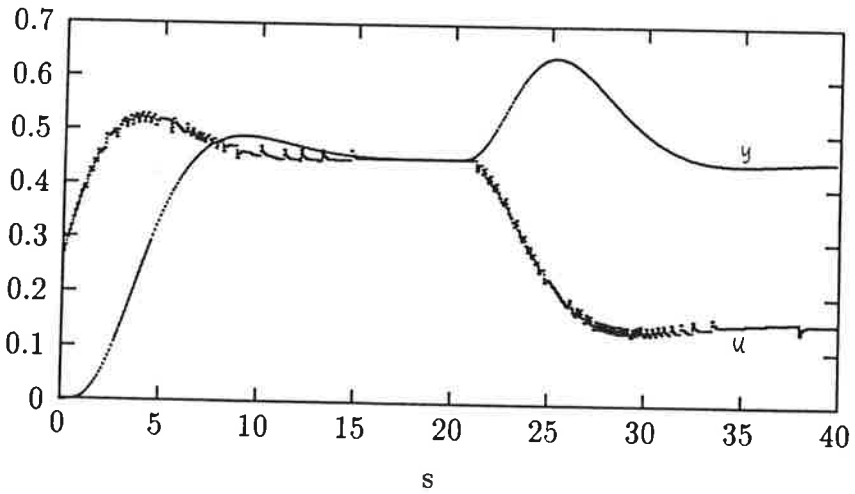


Figure 4.7: Same as 4.6 but with a load disturbance

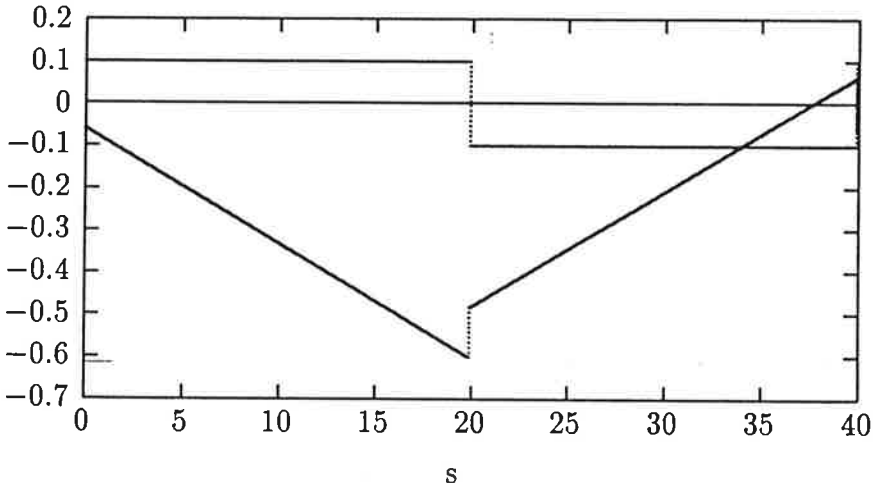


Figure 4.8: Testing proportional and integral action

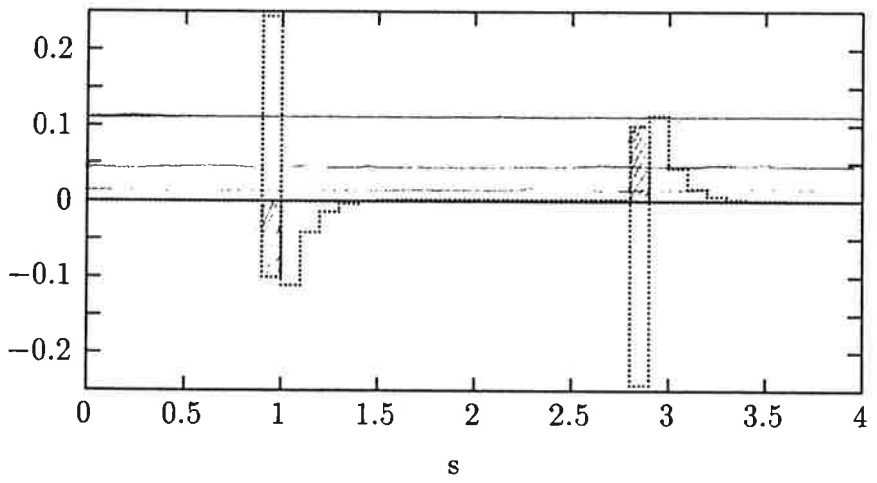


Figure 4.9: Testing derivative action

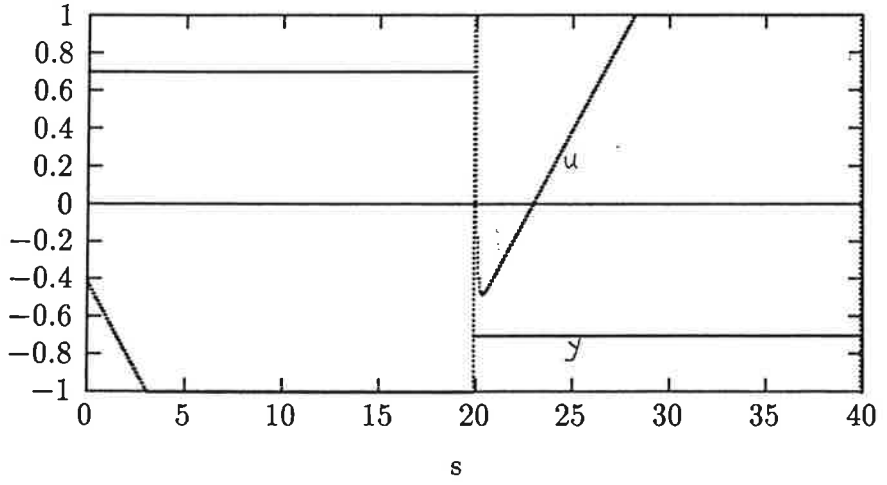


Figure 4.10: Testing saturation arithmetic

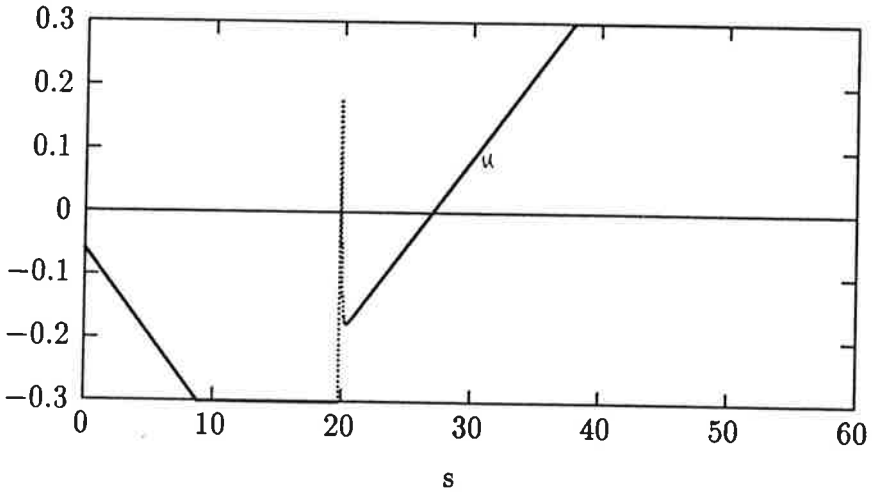
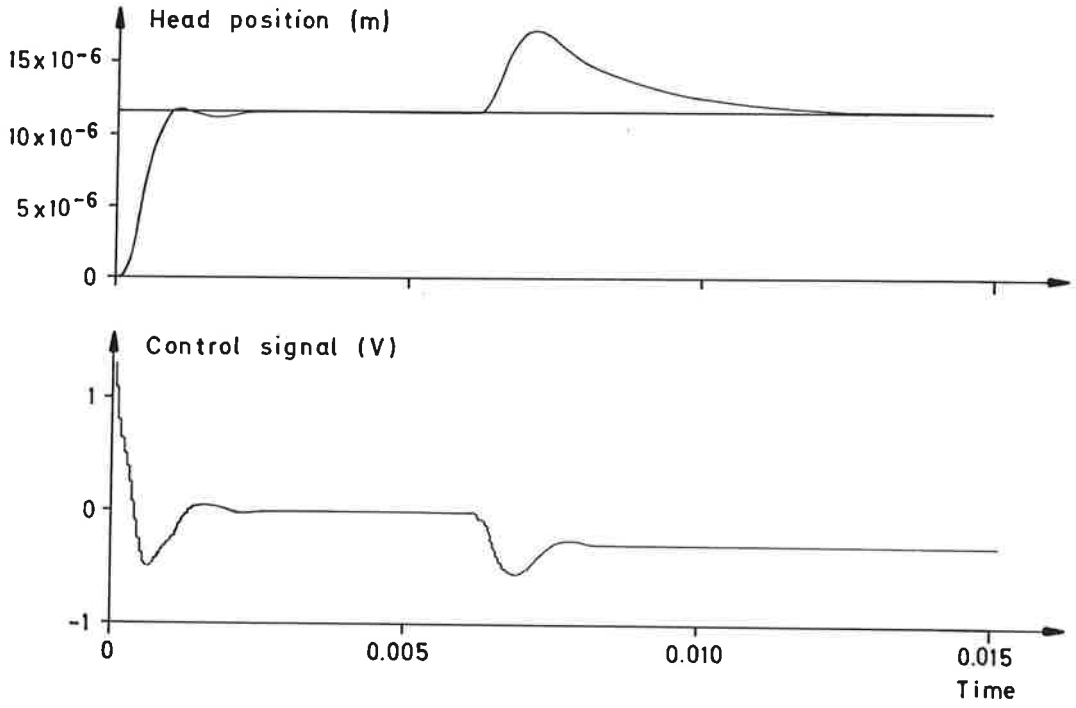


Figure 4.11: Testing anti-windup reset

Fig -orginal till
TFRT-7464

Fig 1



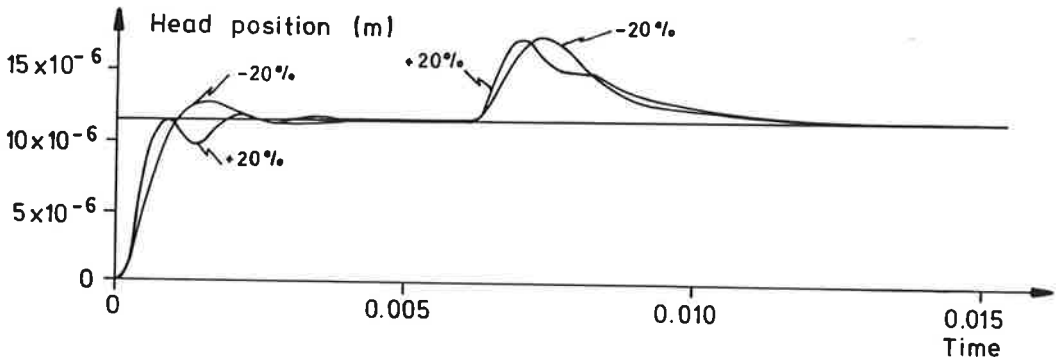


Fig 3

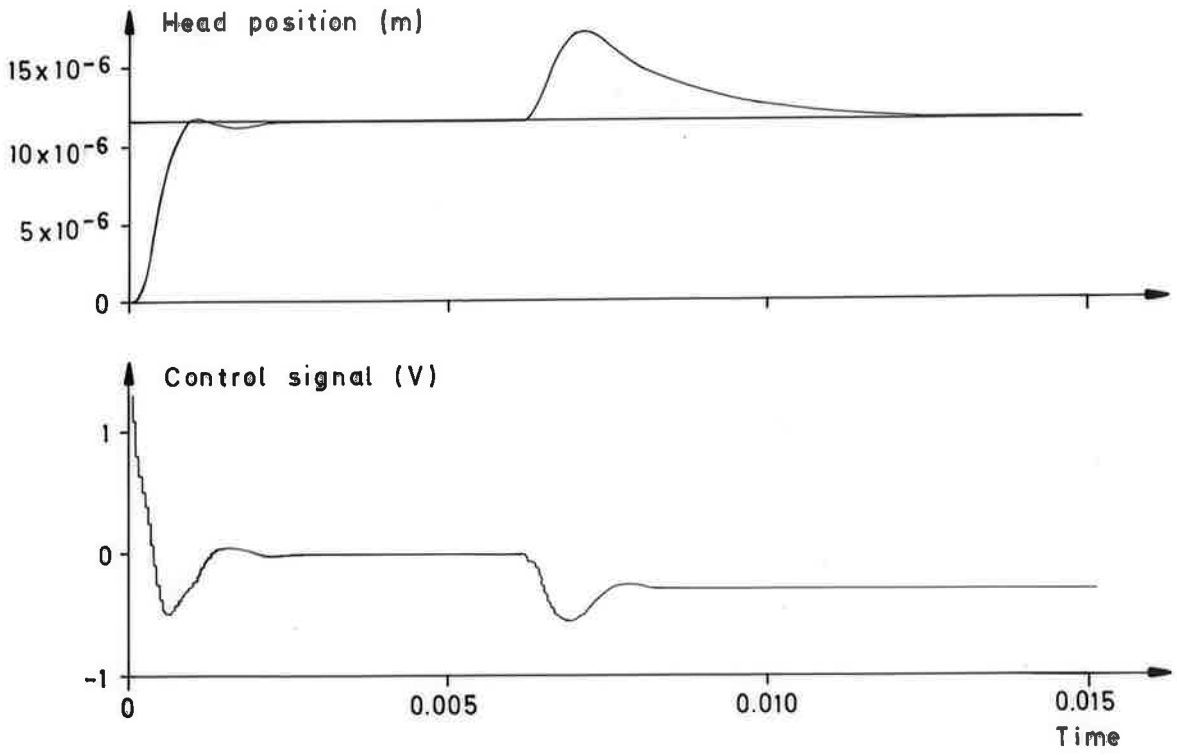

Head position (m)
15x10⁻⁶
10x10⁻⁶
5x10⁻⁶
0
Control signal (V)
1
0
-1
0
0.005
0.010
0.015
Time

Fig 1

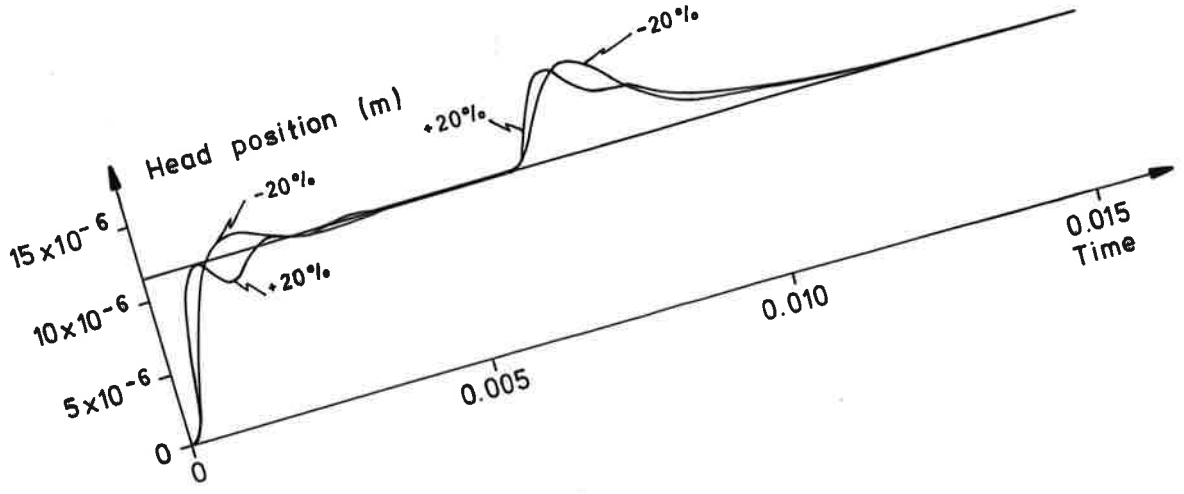

Head position (m)
15x10⁻⁶
10x10⁻⁶
5x10⁻⁶
0
0
0.005
0.010
0.015
Time
-20%
+20%
+20%
-20%

Fig 3

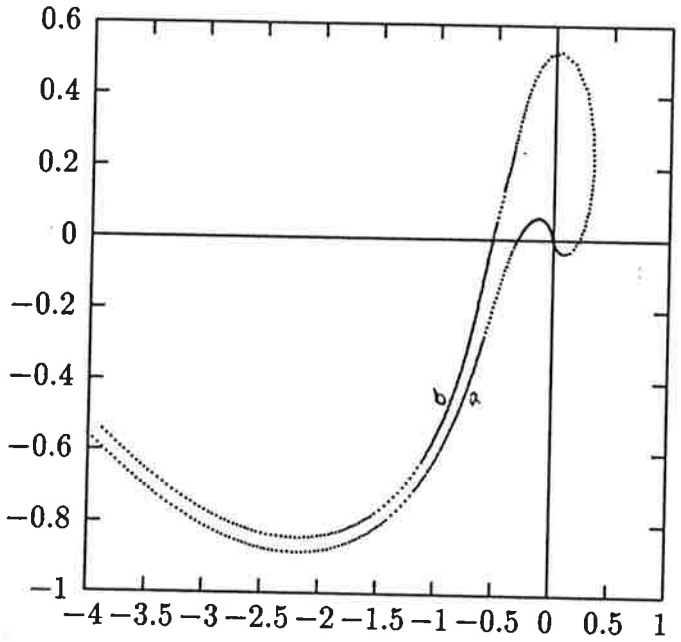


Fig 2

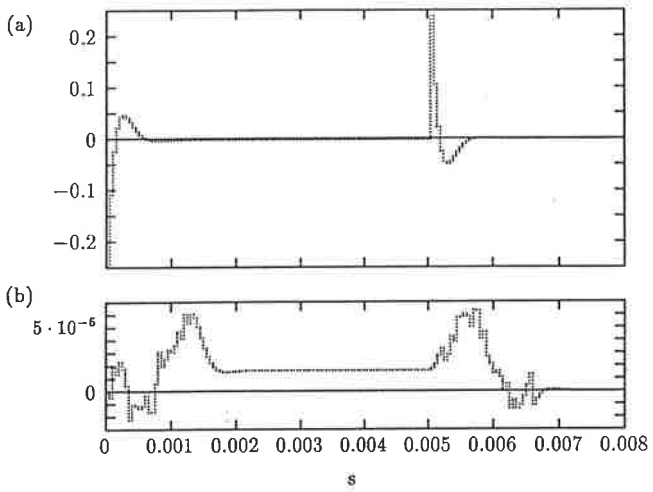

(a)
0.2
0.1
0
−0.1
−0.2
(b)
5 · 10⁻⁶
0
0
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
s

Fig 4

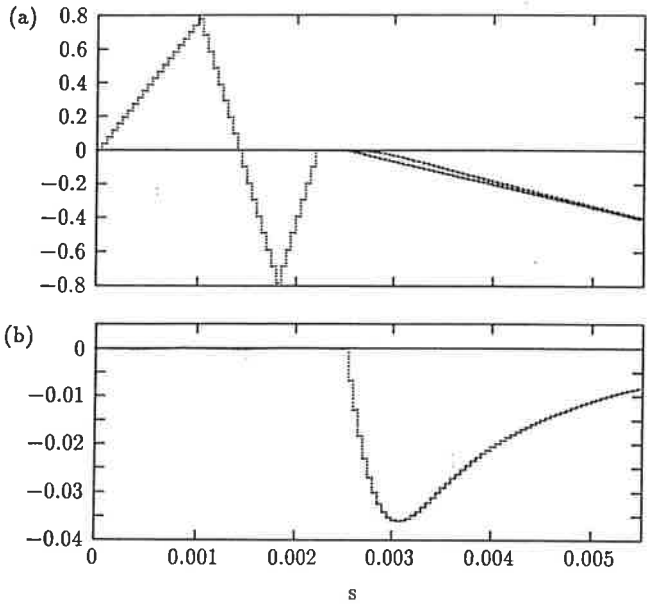

(a)
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
(b)
0
−0.01
−0.02
−0.03
−0.04
0
0.001
0.002
0.003
0.004
0.005
s

Fig 5

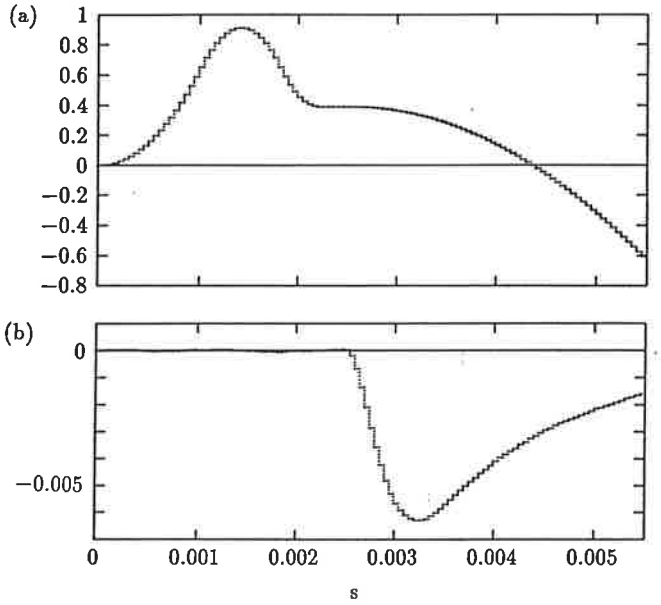


Fig 6

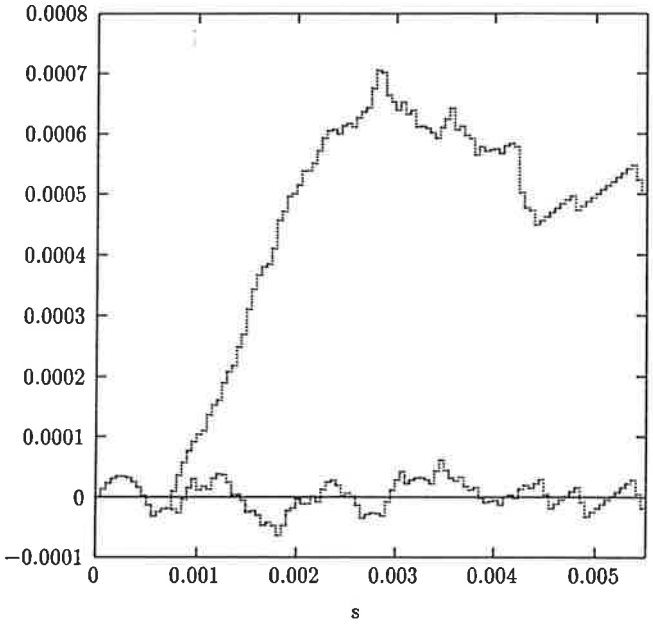

0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0
−0.0001
0
0.001
0.002
0.003
0.004
0.005
s

Fig 7

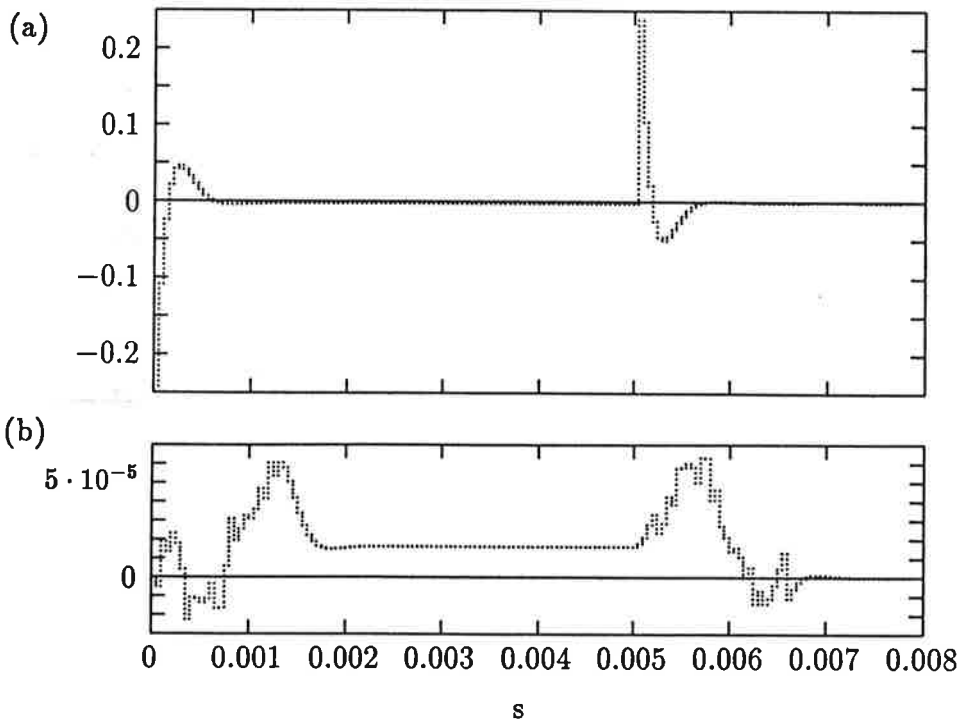

(a)
0.2
0.1
0
−0.1
−0.2
(b)
$5 \cdot 10^{-5}$
0
0
0.001
0.002
0.003
0.004
0.005
0.006
0.007
0.008
s

Fig 4

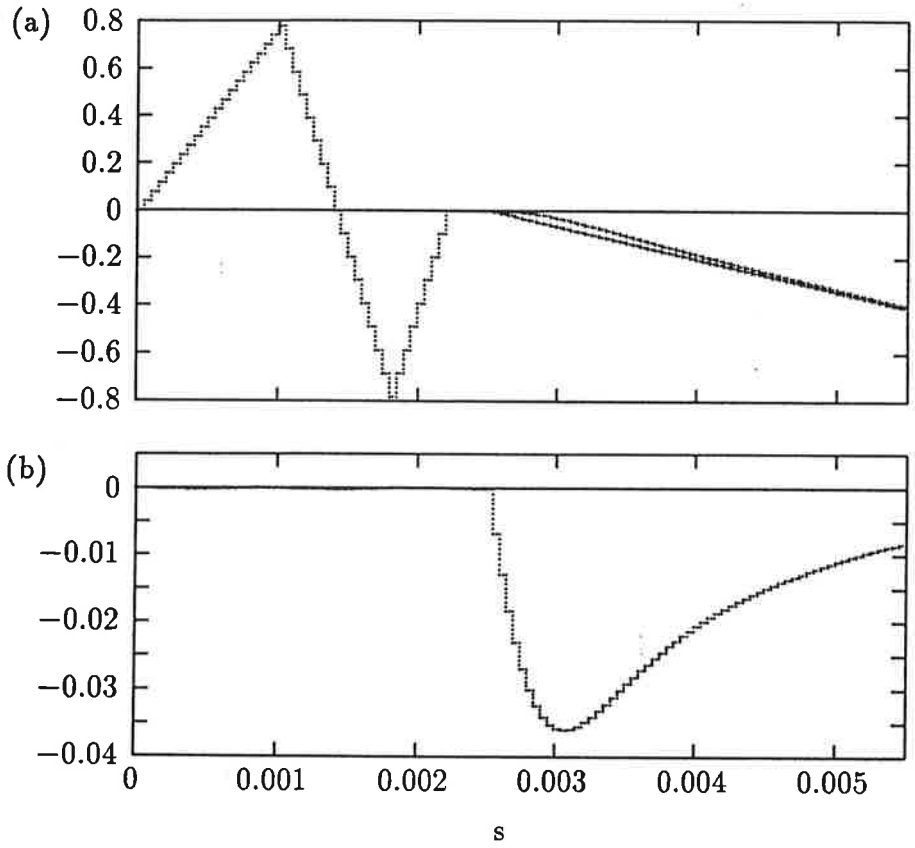

(a)
0.8
0.6
0.4
0.2
0
−0.2
−0.4
−0.6
−0.8
(b)
0
−0.01
−0.02
−0.03
−0.04
0
0.001
0.002
0.003
0.004
0.005
s

Fig 5

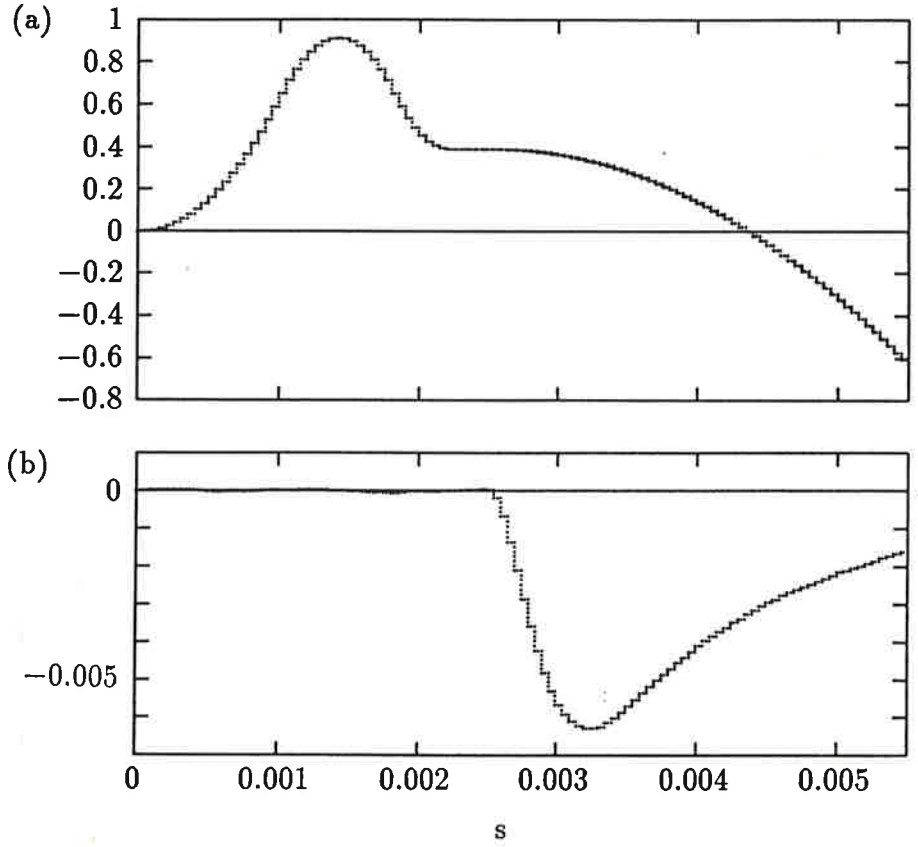


Fig 6

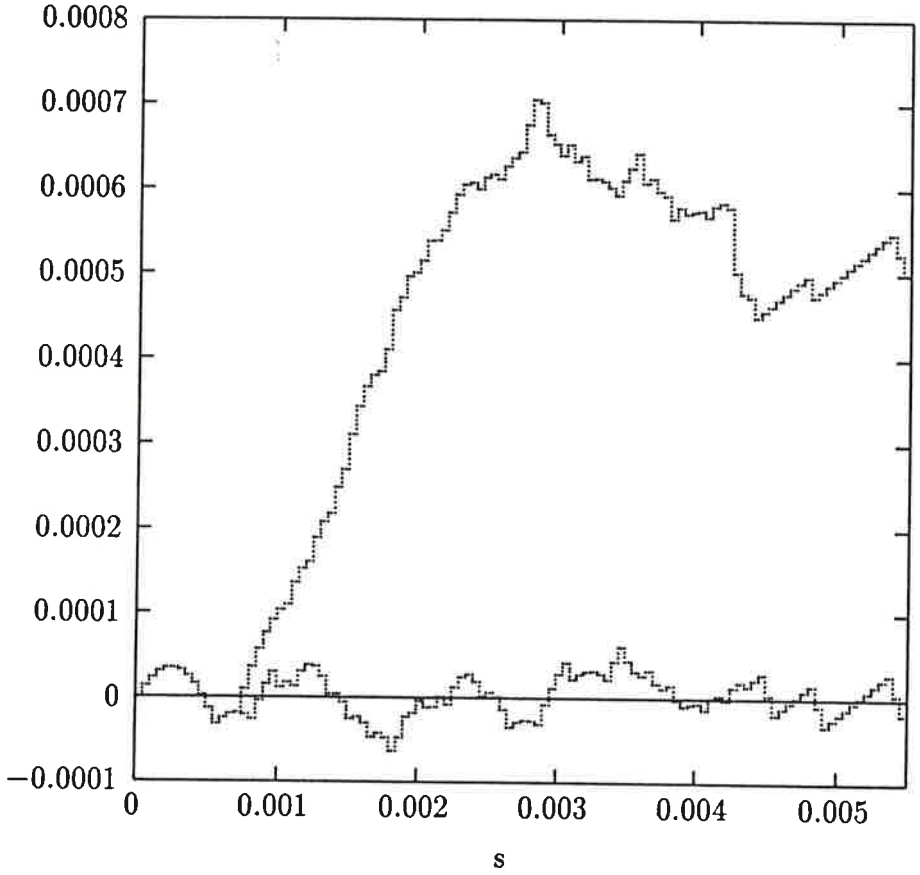

0.0008
0.0007
0.0006
0.0005
0.0004
0.0003
0.0002
0.0001
0
−0.0001
0
0.001
0.002
0.003
0.004
0.005
s

Fig 1