# Implementation of PID Regulators

Åström, Karl Johan

1987

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

CODEN: LUTFD2/(TFRT-7344/1-28/(1987)

# Implementation of PID Regulators

Karl Johan Åström

Department of Automatic Control
Lund Institute of Technology
May 1987

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

*Document name*
Report

*Date of issue*
May 1987

*Document Number*
CODEN: LUTFD2/(TFRT-7344)/1-28/(1987)

*Author(s)*
Karl Johan Åström

*Supervisor*

*Sponsoring organisation*

*Title and subtitle*
Implementation of PID Regulators

*Abstract*

This report treats different issues in the implementation of PID regulators. Both continuous time and discrete time regulators are discussed. A general purpose algorithm which can be used as a building block is proposed.

*Key words*

*Classification system and/or index terms (if any)*

*Supplementary bibliographical information*

*ISSN and key title*

*ISBN*

*Language*
English

*Number of pages*
22

*Recipient's notes*

*Security classification*

# Implementation of PID Regulators

K J Åström

*Abstract*
This report treats different issues in the implementation of PID regulators. Both continuous time and discrete time regulators are discussed. A general purpose algorithm which can be used as a building block is proposed.



Department of Automatic Control
Lund Institute of Technology
Revised May 1987

# 1. Introduction

The purpose of this report is to discuss some practical aspects of PID control in order to arrive at suitable regulator modules which can be used for computer control.

The basic PID algorithm is discussed in Section 2. This section also treats modifications of the proportional and derivative parts to obtain good response to command signals. Different forms of the algorithms, like series parallel and incremental forms are also discussed as well as discretization of the algorithm.

The problem of integrator windup is treated in Section 3. Different ways to avoid windup is treated. The section ends with a proposal for a flexible regulator module that can be used in many different ways.

In Section 4 we discuss different ways to introduce manual control. Particular attention is given to the problem of avoiding transients when switching from manual to automatic mode. A module which is suitable for manual control is introduced.

The results are summarized in Section 5. A few additional modules which can be combined with the PID module and the manual control module are also introduced.

Some special issues associated with digital implementation are discussed in Section 6.

# 2. The PID Algorithm

The basic PID algorithm can be expressed as follows

$$u(t) = k\left(e(t) + \frac{1}{T_i}\int^t e(s)ds + T_d\frac{de}{dt}\right) \tag{2.1}$$

where $u$ is the control variable and $e$ is the control error $e = r - y$ which is the difference between the set point $r$ and the measured value $y$. The control variable is thus the sum of three terms called proportional, integral and derivative action.

In practical regulators the algorithm (2.1) is often modified.

### Proportional Action

It is advantageous to modify the proportional term to

$$P = k(br - y) \tag{2.2}$$

where $b$ is a constant. This modification can be used to reduce the overshoot to step changes in the command signal.

### Derivative Action

The derivative action is often modified to

$$D = -k\frac{pT_d}{1 + pT_d/N}y \tag{2.3}$$

where $p = d/dt$ is the differential operator. This means that the derivation action operates on the output $y$ and not on the command signal. The other modification is that the derivative action only operates on low frequency components. At high frequencies the derivative gain is limited to $kN$.

### Series Form

The algorithm (2.1) is called the parallel form because it can be viewed as a parallel connection of proportional, integral and derivative action. An alternative form is

$$G(s) = k'(1 + \frac{1}{sT_i'})(1 + sT_d') \tag{2.4}$$

This is called the series form because it can be interpreted as a series connection of a PD part with a PI part. The regulator (2.4) can always be represented in the form (2.1) with the coefficients

$$\begin{aligned} k &= k'\frac{T_i' + T_d'}{T_i'} \\ T_i &= T_i' + T_d' \\ T_d &= \frac{T_i'T_d'}{T_i' + T_d'} \end{aligned} \tag{2.5}$$

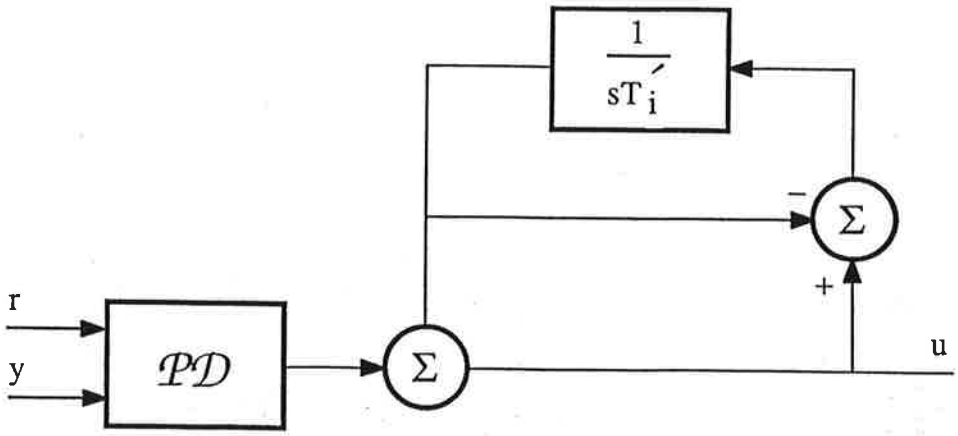


**Figure 2.1** Block diagram of a PID regulator on series form.

The parallel form (2.1) can be transformed to the series form only if

$$T_i \geq 4T_d$$

The parameters are then given by

$$\begin{aligned} k' &= \frac{k}{2}\left(1+\sqrt{1-\frac{4T_d}{T_i}}\right) \\ T_i' &= \frac{T_i}{2}\left(1+\sqrt{1-\frac{4T_d}{T_i}}\right) \\ T_d' &= \frac{T_i}{2}\left(1-\sqrt{1-\frac{4T_d}{T_i}}\right) \end{aligned} \tag{2.6}$$

The parallel form is thus more general than the series form. There is, however, a very simple implementation of the series form which is used in many systems. This implementation is shown in Figure 2.1. The advantages are that it is easy to avoid windup and bumps at mode changes.

### Discretization

To implement a continuous time control law like a PID regulator on a digital computer it is necessary to approximate the derivative and the integral which appear in the control law. A few different ways to do this will now be discussed.

### Proportional Action

The proportional term is

$$P = k(br - y)$$

This term is implemented simply by replacing the continuous variables by their sampled versions. Hence

$$P(t_k) = k(t_k)\,(br(t_k) - y(t_k)) \tag{2.7}$$

where $\{t_k\}$ denote the sampling instants, i.e. the times when the computer reads the analog input.

### Integral Action

The integral term is given by

$$I(t) = \frac{k}{T_i}\int^{t} e(s)ds$$

It thus follows that

$$\frac{dI}{dt} = \frac{k}{T_i}e$$

Approximating the derivative by a difference we get

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

where the sampling period $h = t_{k+1} - t_k$ is assumed constant. This leads to the following recursive equation for the integral term

$$I(t_{k+1}) = I(t_k) + \frac{kh}{T_i}e(t_k) \tag{2.8}$$

For the implementation it is crucial that the wordlength used is sufficiently large so that the term $khe(t_k)/T_i$ is not rounded off when added to $I(t_k)$. This condition is most critical when $h$ is small and $T_i$ is large.

### Derivative Action

The derivative term is given by (2.3) i.e.

$$\frac{T_d}{N}\frac{dD}{dt} + D = -kT_d\frac{dy}{dt} \tag{2.9}$$

There are several ways of approximating the derivative.

### Forward Differences

Approximating the derivative by a forward difference gives the equation

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

This can be rewritten as

$$D(t_{k+1}) = \left(1 - \frac{hN}{T_d}\right)D(t_k) - kN\left(y(t_{k+1}) - y(t_k)\right) \tag{2.10}$$

The approximation is stable only if $T_d > 2Nh$.

### Backward Differences

If the derivatives in (2.9) are approximated by backward differences we get

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

This can be rewritten as

$$D(t_k) = \frac{T_d}{T_d + Nh}D(t_{k-1}) - \frac{kT_dN}{T_d + Nh}\left(y(t_k) - y(t_{k-1})\right) \tag{2.11}$$

The approximation is stable for all positive $T_d$.

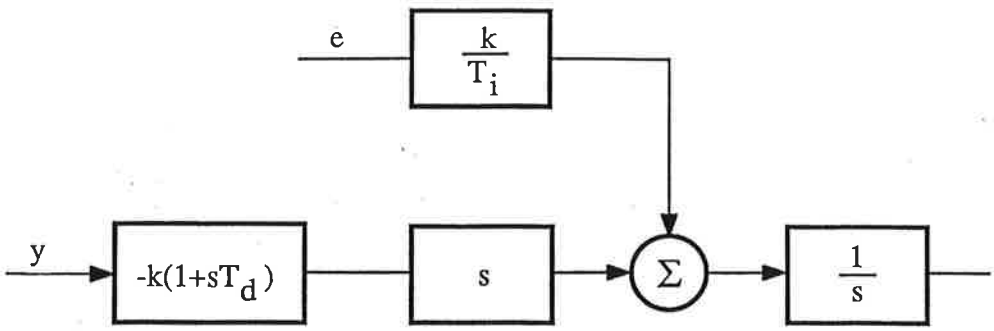


**Figure 2.2** Block diagram of an incremental PID algorithm.

### Tustin's Approximation

There is yet another approximation proposed by Tustin which is commonly used. This approximation is

$$D(t_k) = \frac{2T_d - hN}{2T_d + hN} D(t_{k-1}) - \frac{2kNT_d}{2T_d + hN} (y(t_k) - y(t_{k-1})) \tag{2.12}$$

Notice that all approximations have the same form i.e.

$$D(t_k) = a_i D(t_{k-1}) - b_i (y(t_k) - y(t_{k-1})) \tag{2.13}$$

but with different values of the parameters $a_i$ and $b_i$. The approximation (2.12) is stable if $T_d > 0$. The value of $a_i$ is, however, negative if $T_d < Nh/2$. This is undesirable because the approximation will then exhibit ringing. This effect can be very significant if $T_d \ll Nh$. Hence only the approximation (2.11) gives good results for all values of $T_d$.

### Incremental Form

The algorithms described so far are called positional algorithms because they give the output of the regulator directly. In digital implementations an incremental form of the algorithms is also used. This form is obtained by computing the time differences of the regulator output and adding the increments. In a continuous time version the time derivative of the output is computed and the derivative is then integrated. A block diagram of an incremental algorithm is shown in Figure 2.2. This form is particularly useful when the actuator is a stepping motor because the motor can then be used as the summing device.

One advantage with the incremental algorithm is that most of the computations are done using increments only. Short wordlength calculations can often be used. It is only in the final stage where the increments are added that precision is needed. Another advantage with the incremental algorithm is that the regulator output is driven directly from an integrator. This makes it very easy to deal with windup and manual control. A problem with the incremental algorithm is that it can not be used directly for regulators with $P$ or $PD$ action only. Such a regulator cannot keep a proper steady state because the output of the regulator depends on the initial state of the regulator. See

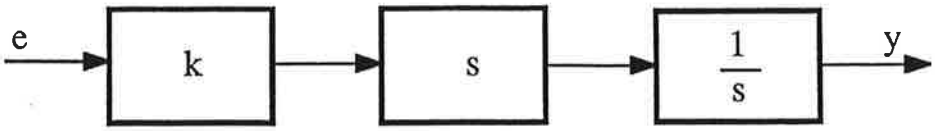


**Figure 2.3** An incremental form for a proportional regulator which does not work.

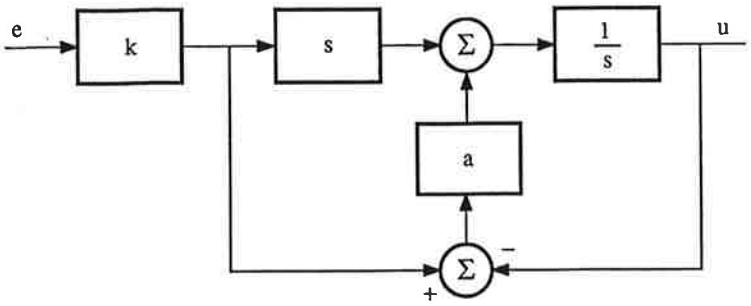


**Figure 2.4** An incremental form of a proportional regulator which works.

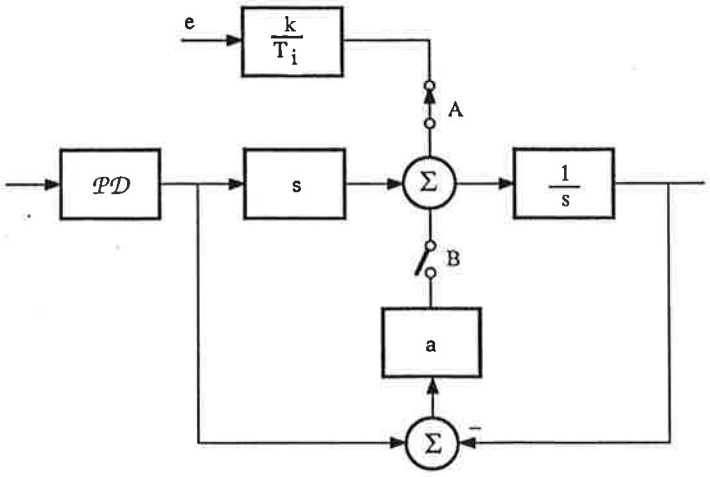


**Figure 2.5** Block diagram of a PID regulator on incremental form. The switch B is open and switch A is closed when there is integral action. The switch A is open and B is closed when there is no integral action.

Figure 2.3. The problem can be avoided with the regulator shown in Figure 2.4 which contains a feedback that resets the integrator to a proper value. A PID regulator on incremental form can be obtained by combining the systems in Figure 2.2 and 2.4 as shown in Figure 2.5.

# 3. Windup

Although many aspects of a control system can be understood based on linear theory there are some nonlinear effects that must be accounted for. All actuators have limitations, a motor has limited speed, a valve cannot be more than fully open or fully closed etc. When a control system operates over a wide range of operating conditions it may happen that the control variable reaches the actuator limits. When this happens the feedback loop is effectively broken because the actuator may remain at its limit independently of the process output. If a regulator with integrating action is used, the error may continue to be integrated. This means that the integral term may become very large or colloquially that it "winds up". The consequence is that any regulator with integral action may give large transients when the actuator saturates.

### An Example

The windup phenomena is illustrated in Figure 3.1 which shows control of a process with a PI regulator. The initial set-point change is so large that the actuator saturates at the high limit. The integrator increases initially because the error is positive and it reaches its largest value at time $t = 10$ when the error goes through zero. The output remains saturated at this point because of the large value of the integral. It does not leave the saturation limit until the error has been negative for sufficiently long time to let the integral part come down to a small level. The net effect is a large overshoot which is clearly noticeable in the figure.

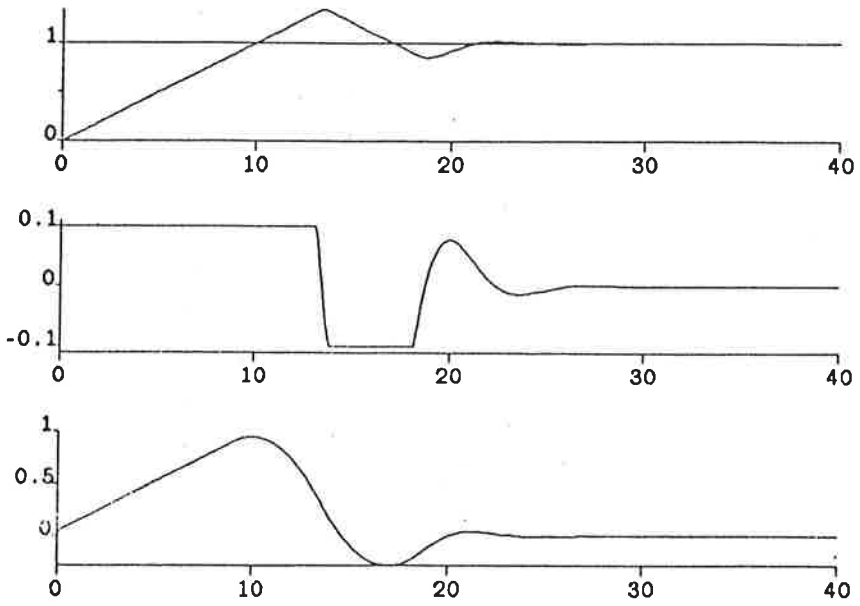


**Figure 3.1** Illustration of integral windup.

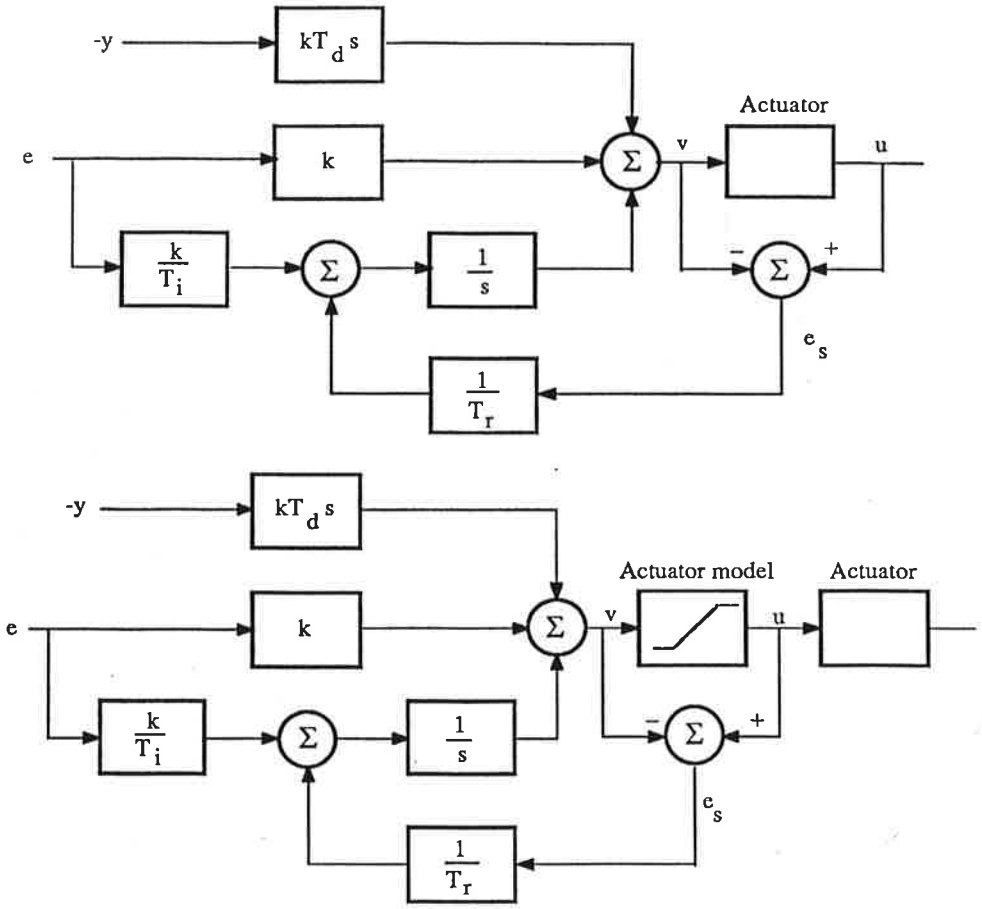


**Figure 3.2** Regulator with anti-windup. A system where the actuator output is measured is shown in A and a system where the actuator output is estimated from a mathematical model is shown in B.

### How to Avoid it

There are several ways to avoid integral windup. A convenient way is shown in Figure 3.2. An extra feedback path is provided in the regulator by measuring the actual actuator output and forming an error signal $e_s$ as the difference between the output of the regulator $v$ and the actuator output $u$. The signal $e_s$ is fed to the input of the integrator through a gain $1/T_r$. The signal $e_s$ is zero when there is no saturation. It will thus not have any effect on the normal operation when the actuator does not saturate. When the actuator saturates the feedback signal will, however, drive the error $e_s$ to zero. This means that it drives the integrator to a value such that the regulator output is exactly at the saturation limit. This will clearly prevent the integrator from winding up. The rate at which the regulator output is reset is governed by the feedback gain $1/T_r$, where $T_r$ can be interpreted as the time constant which determines how quickly the integral is reset.

It frequently happens that the actuator output cannot be measured. The anti-windup scheme just described can be applied by incorporating a mathematical model of the saturating actuator as is illustrated in Figure 3.2.

Figure 3.3 shows what happens when a regulator with anti-windup is applied to the system simulated in Figure 3.1. Notice that the output of the integrator is quickly reset to a value such that the regulator output is at the saturation limit and that the integral has a negative value during the initial phase when the actuator is saturated. This behavior is drastically different from that in Figure 3.1 where the integral has a positive value during the initial

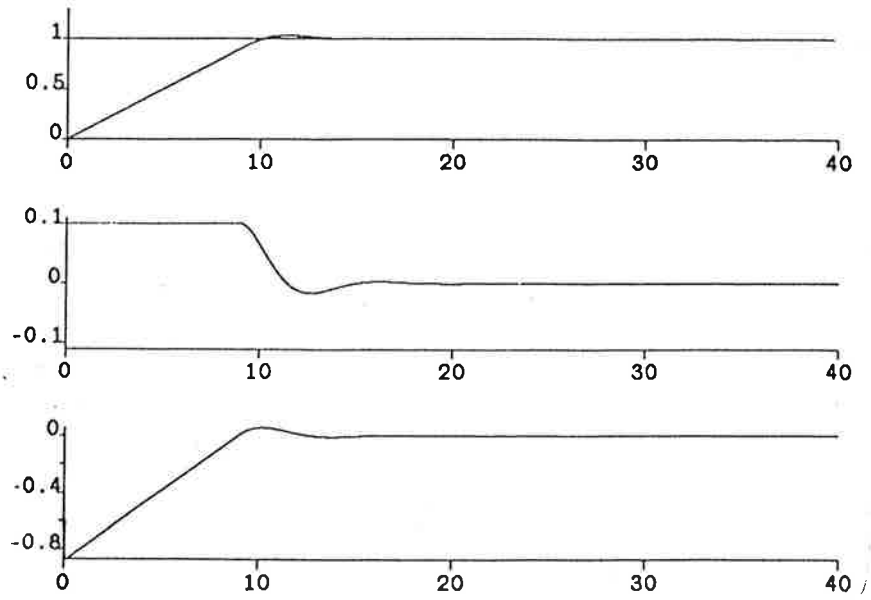


**Figure 3.3** Regulator with anti-windup applied to the system in Figure 3.1.

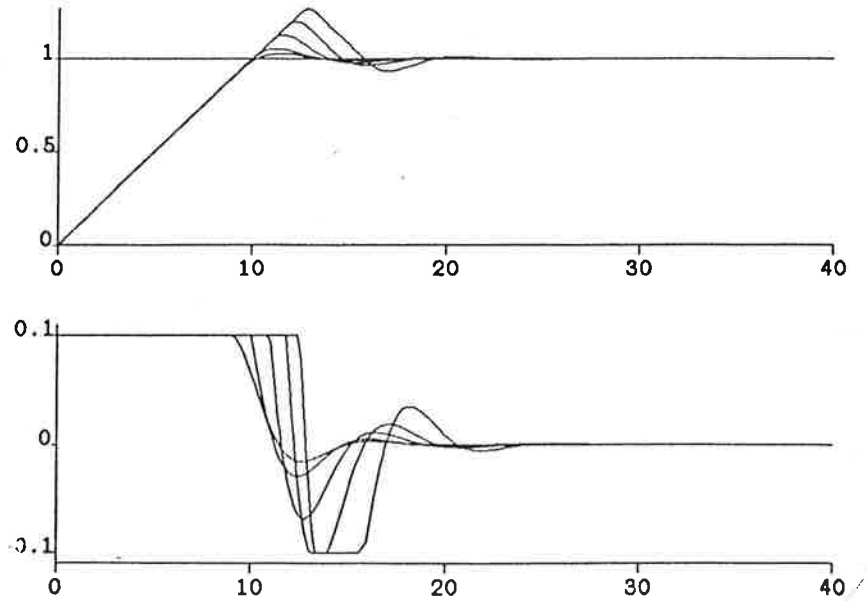


**Figure 3.4** The step response of the system in Figure 3.3 for different values of the reset time constant $T_r$.

transient. Also notice the drastic improvement in performance compared to the ordinary PI regulator used in Figure 3.1.

The effect of different values of the time constant $T_r$ is illustrated in Figure 3.4. It may thus seem advantageous to always choose a very small value of the time constant $T_r$ because the integrator is then reset quickly. Some care must, however, be exercised when introducing anti-windup in systems with derivative action. If the time constant $T_r$ is chosen too small it may happen that spurious errors cause saturation of the output due to a large derivative term, and this may accidentally reset the integrators to a strange value. A practical rule is to make $T_r$ proportional to the integration time $T_i$.

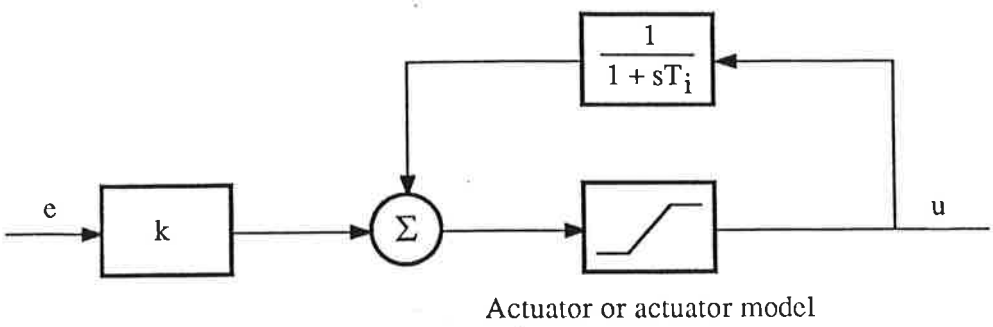


**Figure 3.5** How to provide anti-windup in a regulator where integral action is generated as automatic reset.

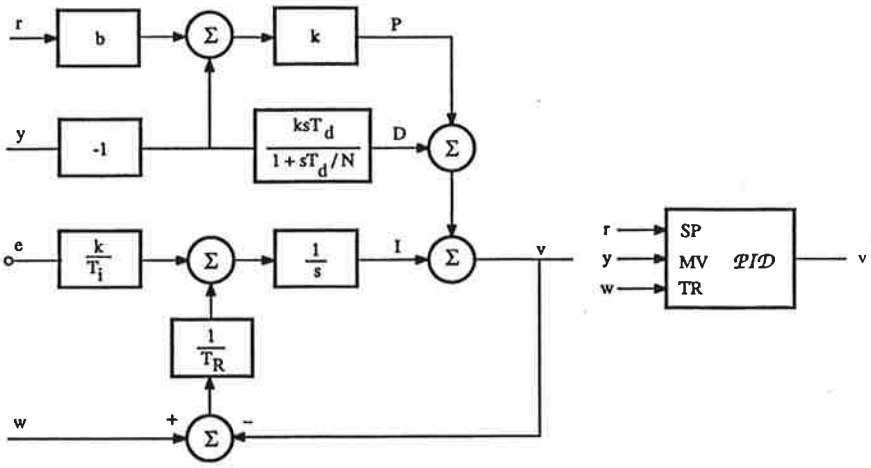


**Figure 3.6** Block diagram and simplified representation of PID regulator with tracking signal.

### Series Implementation

A similar device for avoiding windup can be applied to the regulator in Figure 2.1 by incorporating a model of the saturation as is shown in Figure 3.5. Notice that in this implementation the reset time constant $T_r$ is the same as the integration time $T_i$.

### A Regulator Module

The systems shown in Figure 3.2 can be conveniently represented if we introduce the module shown in Figure 3.6. The module has three inputs, the set point, the measured output and a tracking signal. The new input TR is called a tracking signal because it follows from Figure 3.6 that the regulator output $v$ tries to track this signal. Using such a model the systems shown in Figure 3.2 can be presented as shown in Figure 3.7. The parameters are the PID parameter ($k, T_i, T_d, b$ and $N$) and the reset time constant $T_r$.

### Systems with Selectors

A selector is a device with several inputs and one output. The output is at each time the smallest of the inputs for a minimum selector or the largest input for a maximum selector. Selectors are used to make sure that constraints are satisfied.

When selectors are used to choose among the outputs of several regulators with integral action it is crucial that anti-windup is considered. This is easily

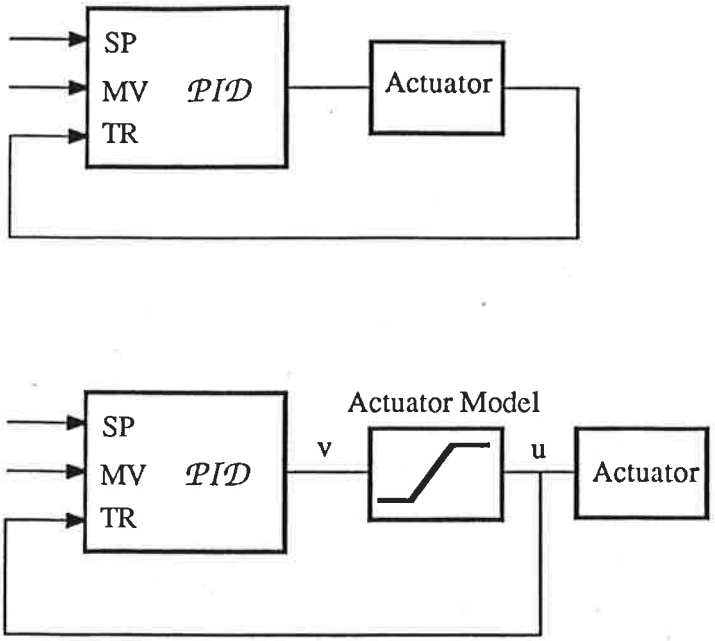


**Figure 3.7** Representation of the regulators with anti-windup in Figure 3.2 using the basic control module with tracking shown in Figure 2.6.

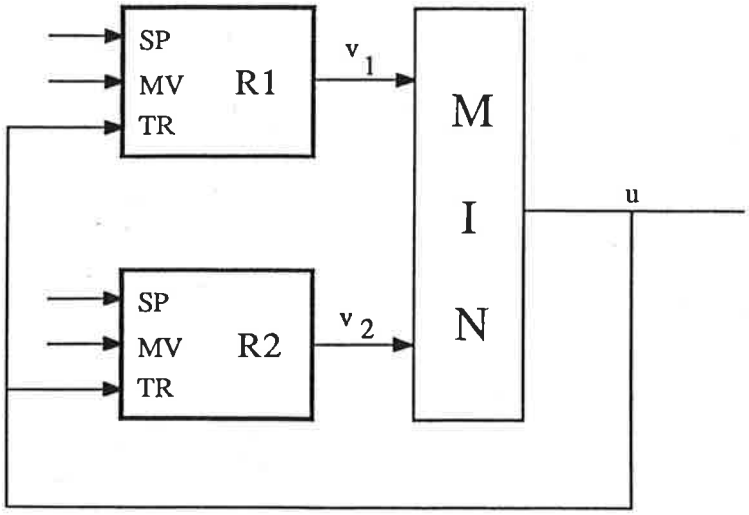


**Figure 3.8** How to avoid windup in circuits with selectors.

handled using the regulator module with a tracking input. Figure 3.8 shows how the regulators can be connected. When $v_1 < v_2$ the output is $u = v_1$. The output $u$ is thus controlled by regulator $R_1$. The regulator $R_2$ will track $u$ since $v_2 \neq u$. A simulation of such a scheme is shown in Figure 3.9.

### Cascade Control

Avoiding windup in cascade control poses special problems. Windup for the secondary regulator can be handled in the usual manner. To avoid windup in the primary regulator it is, however, necessary to know that the secondary regulator saturates. One strategy is to put the primary regulator into manual control when the secondary regulator saturates.

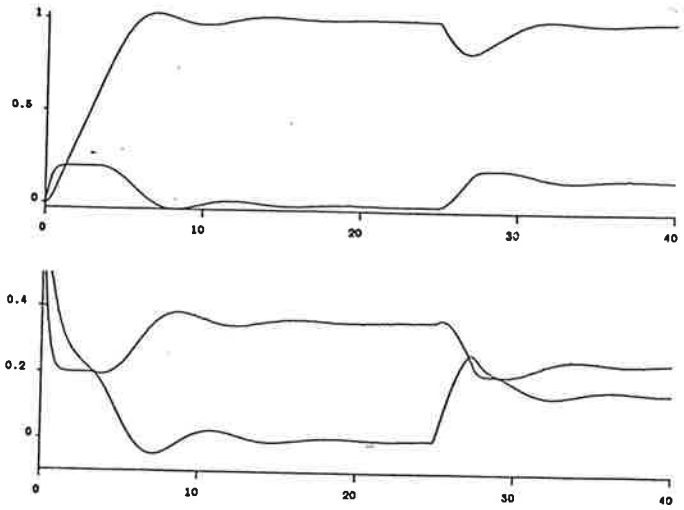


**Figure 3.9** Simulation of a system with selectors.

# 4. Manual Control

Most control systems need a facility to be run under manual control. To achieve this it is necessary to have a convenient way to switch off the automatic control action and to change the control variable of the process directly. The manual control is often done using two buttons. The control variable increases when pushing one, it decreases when the other one is pushed. The control variable remains constant if neither button is pushed. A facility of this type is provided even in the most simple regulators. There is typically a mode switch for manual and automatic and increase/decrease buttons. It is of course also necessary to have a smooth transfer between the manual mode and the automatic mode. Since the command buttons only give the changes in the control variables it is necessary to have an internal state which represents the sum of the changes. To ensure a smooth transfer between the manual and automatic modes it is necessary to ensure that the state associated with manual control is updated properly when the regulator is in automatic mode and vice versa.

### Incremental Algorithms

A bumpless switch between automatic and manual is particularly easy to do in incremental algorithms when the control variable is driven directly by an integrator. The integrator is provided with a switch so that either the increments from the manual control input or the increments from the PID algorithms are sent to the integrator. See Figure 4.1.

### Absolute Algorithm with Series Implementation

A similar mechanism can be used in the series implementation of a PID controller shown in Figure 2.1. See Figure 4.2. In this case there will be a switching transient if the output of the PD part is not zero at the switching instant. Notice that it is necessary to have two switches.

### Parallel Implementation

For regulators with parallel implementation the integrator of the PID regulator can be used to add up the changes in manual mode. The regulator shown in Figure 4.3 is such a system. This system gives a smooth transition between manual and automatic mode provided that the switch is made when output of the PD block is zero. If this is not the case there will be a switching transient.

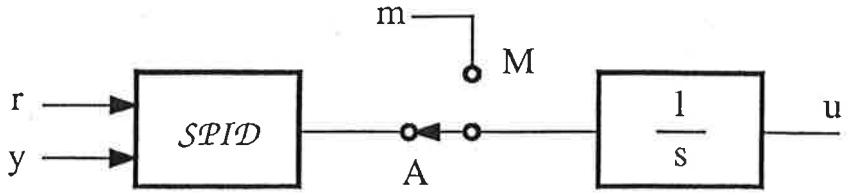


**Figure 4.1** How to introduce manual control in a regulator with incremental output.

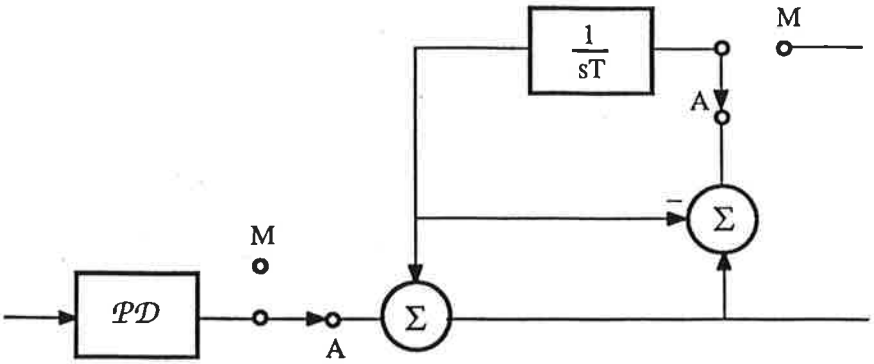


**Figure 4.2** How to introduce manual control in a PID regulator with a special series implementation.

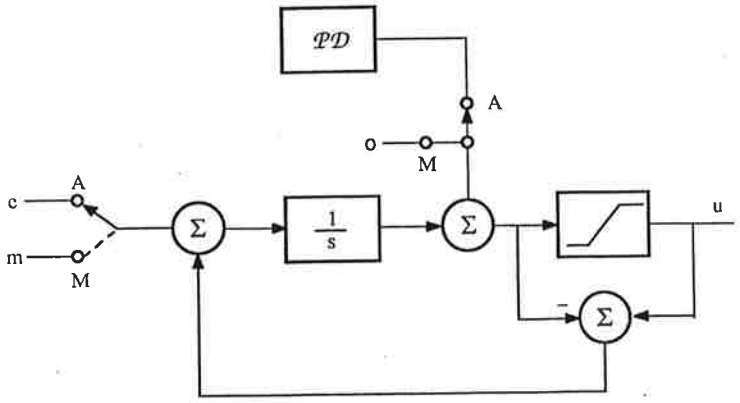


**Figure 4.3** A PID regulator where one integrator is used both to obtain integral action in automatic mode and to sum the incremental commands in manual mode.

This will almost always be the case when the PD action is given by

$$P + D = k\left(br - y - T_d \frac{dy}{dt}\right)$$

with $b \neq 1$.

It is also possible to use a separate integrator to add the incremental changes from the manual control device. To avoid switching transients in such a system it is necessary to make sure that the integrator in the PID regulator is reset to a proper value when the regulator is in manual mode. Similarly the integrator associated with manual control must be reset to a proper value when the regulator is in automatic mode. This can be realized with the circuit shown in Figure 4.4. With this system the switch between manual and automatic is smooth even if the control error or its derivative is different from zero at the switching instant. When the regulator operates in manual mode as is shown in Figure 4.4 the feedback from the output $v$ of the PID regulator tracks the output $u$. With efficient tracking the signal $v$ will thus be close to $u$ at all times. There is a similar tracking mechanism which ensures that the integrator in the manual control circuit tracks the regulator output in manual mode.

To build large automation systems it is useful to have suitable modules. Figure 4.5 shows the block diagram for the manual control module. It has two inputs, a tracking input and an input for the manual control commands. The

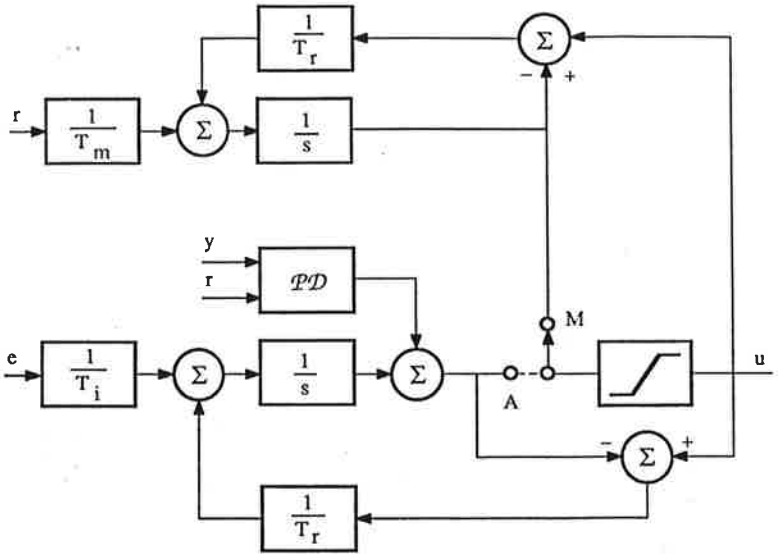


**Figure 4.4** PID regulator with parallel implementation which switches smoothly between manual and automatic control.

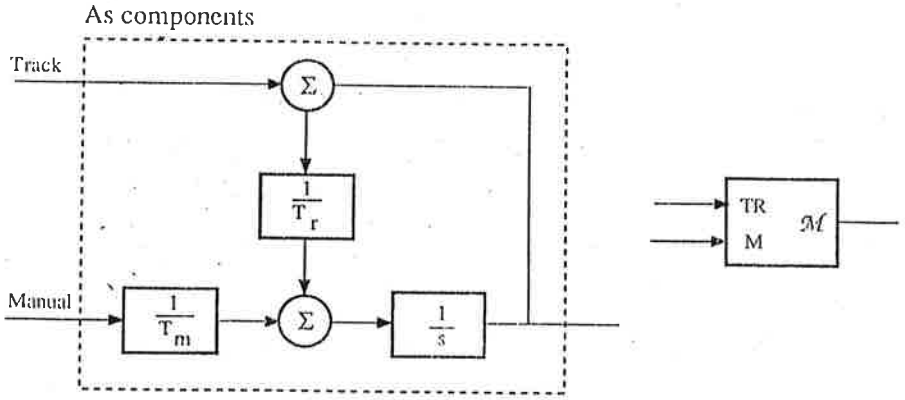


**Figure 4.5** Manual control module.

system has two parameters, the time constant $T_m$ for the manual control input and the reset time constant $T_r$. In digital implementations it is convenient to add a feature so that the command signal accelerates as long as one of the buttons increase-decrease buttons are pushed.

Using the module for PID control, introduced in Figure 3.6, and the manual control module in Figure 4.5 it is straightforward to construct a complete regulator. Figure 4.6 shows a PID regulator with internal or external setpoints via increase/decrease buttons and manual automatic mode. Notice that the system only has two switches.

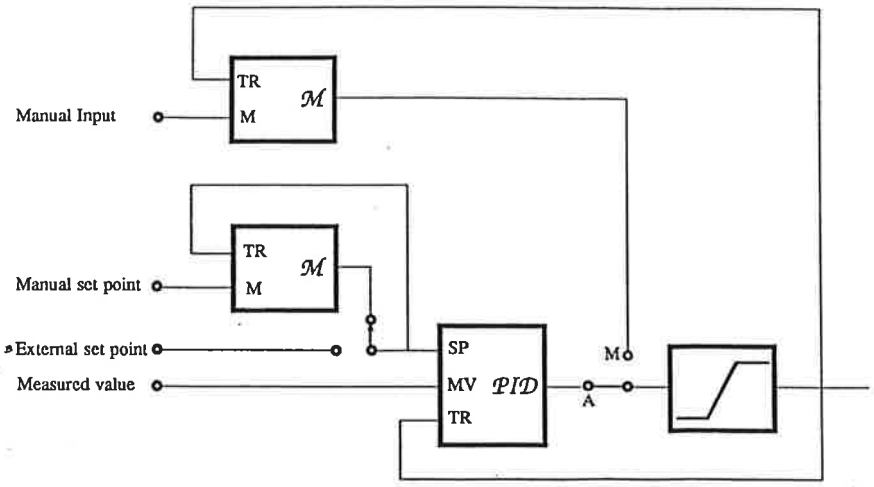


**Figure 4.6** A reasonably complete PID regulator with anti-windup, automatic-manual mode, and manual and external setpoint.

# 5. Some Additional Modules

We have thus arrived at two modules which are useful for building control systems, a manual control module and a PID module. Both modules have internal states and a tracking input. To design complete systems it is also useful to add modules for selection of maximum and minimum. To model actuators we also need a module for saturating a signal. It is sometimes needed to make sure that command signals do not change too rapidly. This can be ensured by using a jump- and rate module. See Figure 5.1. The module has one input and one output and four parameters.

The properties of the jump and rate circuit are illustrated in Figure 5.2. Summarizing we have the following modules.

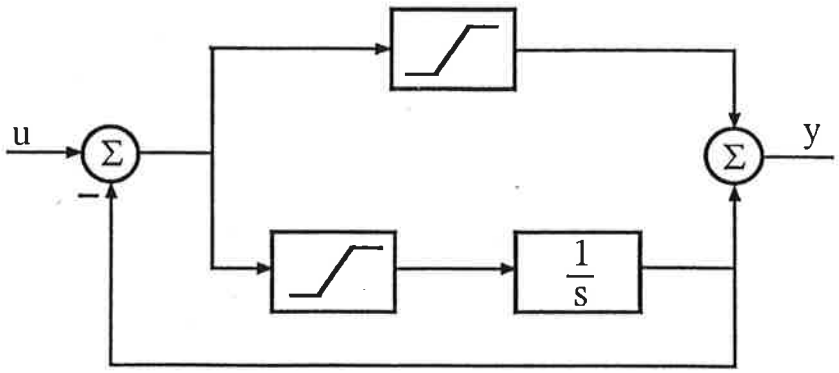


**Figure 5.1** Block diagram of a jump- and rate circuit.

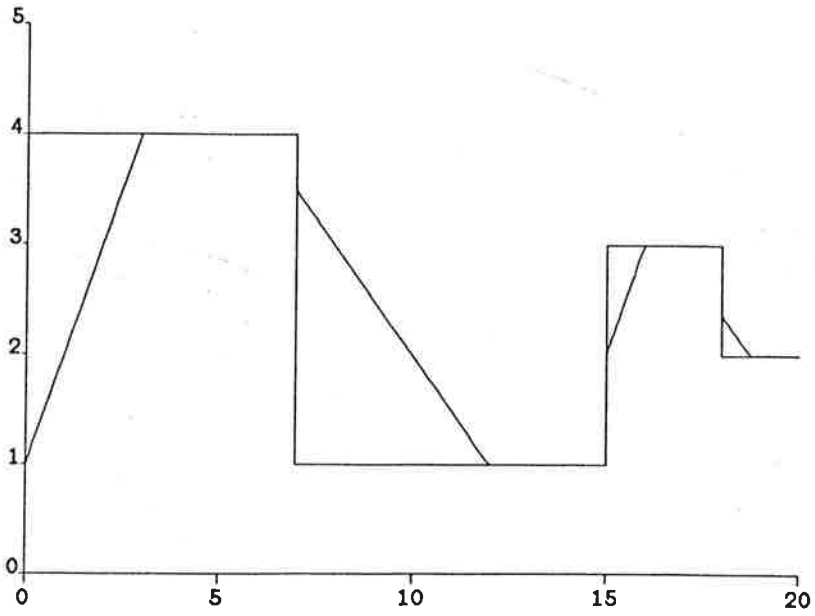


**Figure 5.2** Input and outputs for the jump and rate.

PID
Manual
Saturation
Min selector
Max selector
Jump and rate

A detailed specification of these modules is given in Appendix A which also contains Simnon programs. Using these modules we can now construct large automation systems.

# 6. Digital Implementation

The diagrams shown so far are analog realizations where all operations are executed in parallel. When the control algorithms are implemented on a digital computer the parallel operations have to be realized sequentially. To do this there are especially two problems that must be taken into account, namely simultaneity and time delays.

### Generic Control Law

To discuss the issues a generic form of the control law is first given. A general linear control law can be written as follows

$$u(k) = Cx(k) + Dy(k) + D_c r(k)$$
$$x(k+1) = Fx(k) + Gy(k) + G_c r(k)$$

where $r$ is the command signal, $y$ the measured signal and $u$ the control variable.

### Computational Delay

The control program can then be written as

```
1 Adin y r
2 u:=C*x+D*y+Dc*r
3 x:=F*x+G*y+Gc*r
4 Daout u
```

**Listing 6.1** Improved regulator code.

It is desirable to make the computational delay as small as possible. Notice that the DA conversion can be made after the second statement since the control signal $u$ is then available. Also notice that the product $C * x$ can be precomputed. The code then becomes

```
1 Adin y r
2 u:=u1+D*y+Dc*r
3 ShapeOutput
4 Daout u
5 x:=F*x+G*y+Gc*r
6 u1:=C*x
```

**Listing 6.2** Improved regulator code.

Notice that an extra state variable $u$, has been introduced to save computing $C * x$ in the first statement. Also notice that a procedure ShapeOutput which saturates the output, make a min selector etc. is added.

The code shown above can be generalized as follows

```
1 Adin y r
2 ComputeOutput
3 ShapeOutput
4 Daout
5 UpdateState
```

**Listing 6.3** Generalized regulator code.

It would be appealing to make a procedure for each one of the boxes PID, Manual etc. in the Figures 3.7, 3.8, 4.5 and 4.6. This cannot be done because of the sequential character of the calculations.

In the analog implementation the signals $v$ and $u$ will change simultaneously. This is essential for the antiwindup signal to function properly. In a digital implementation there will always be a delay between $u$ and $v$. If the PID regulator and the actuator are implemented as separate blocks there is no way to avoid this delay. The tracking signal $u$ will then differ from $v$ and the antiwindup coupling will give an undesired contribution. Although this contribution will be small if the delay is small it is always present. This undesirable effect can be avoided if the code is restructured.

There is also another problem if the blocks in Figure 3.7 are represented as separate subroutines. If the antiwindup scheme should work properly it is essential that simultaneous values of the tracking signal $t_r$ and the regulator output $v$ are used. This will not be the case if the regulator code is executed first and the actuator model afterwords.

It is thus essential that the computational scheme shown in Listing 6.3 is used in digital implementations. To obtain the appropriate structure the algorithms for the discrete time PID regulator will be rewritten appropriately.

The regulator output is given by

$$u(t_k) = P(t_k) + I(t_k) + D(t_k) \tag{6.1}$$

where P, I and D are given by the equations (2.7), (2.8) and (2.9) respectively. It follows from (2.7) that the proportional part cannot be precomputed. Equation (2.8) shows that the integral term can be precomputed. For this purpose we introduce $I$ as a state variable. It follows from equation (2.13) that part of the derivative term can be precomputed. A state variable $x$ is introduced to account for those terms. This state variable is defined as

$$x(t_k) = a_i D(t_{k-1}) - b_i y(t_{k-1}) \tag{6.2}$$

The derivative term then becomes

$$D(t_k) = x(t_k) - b_i y(t_k) \tag{6.3}$$

It follows from (6.2) and (6.3) that the state variable $x$ is updated as follows

$$\begin{aligned} x(t_{k+1}) &= a_i\left[x(t_k) - b_i y(t_k)\right] - b_i y(t_k) \\ &= a_i x(t_k) - b_i(1 + a_i) y(t_k) \end{aligned} \tag{6.4}$$

Equation (6.1) can now be written as

$$\begin{aligned} u(t_k) &= bkr(t_k) - \left[k + b_i\right] y(t_k) + I(t_k) + x(t_k) \\ &= \ell_0 r(t_k) - \ell y(t_k) + u_1(t_k) \end{aligned} \tag{6.5}$$

and the equations for updating the states becomes

$$\begin{aligned} I(t_{k+1}) &= I(t_k) + \frac{kh}{T_i}\left[r(t_k) - y(t_k)\right] \\ x(t_{k+1}) &= a_i x(t_k) - b_i(1 + a_i) y(t_k) \\ u_1(t_{k+1}) &= I(t_{k+1}) + x(t_{k+1}) \end{aligned} \tag{6.6}$$

The procedure ComputeOutput in Listing 6.3 is then an implementation of (6.4) and the procedure UpdateState is a procedure which performs the calculations given by (6.6).

# Appendix A

## Simnon Programs for the Figures

```
MACRO fig31
"Simulation of integrator windup
SYST  int pid conn
PAR N:0
PAR Ti:1
PAR b:1
PAR ulow:-1
PAR uhigh:1
AXES h 0 40 v 0 1.5
PLOT y[proc] r
STORE y[proc] u[proc] r i up
SIMU 0 40
SPLIT 3 1
ASHOW y r
AXES V -0.11 0.11
SHOW up
ASHOW i

END


CONTINUOUS SYSTEM proc
"Integrator with saturation
"File called int

INPUT u
OUTPUT y
STATE x
DER dx

y=x
up=if u<ul then ul else if u<uh then u else uh
dx=up

ul:-0.1
uh:0.1

END
```

```
CONTINUOUS SYSTEM reg

"PID regulator module. File called pid

INPUT r y v
OUTPUT u
STATE i   x
DER   di dx

"Output
u=P+I+D

"Proportional action
P=k*(b*r-y)

"Derivative action
dx=(y-x)*N/Td
d1=k*N*(x-y)
D=if d1<dlow then dlow else if d1<dhigh then d1 else dhigh

"Integral action
dI=k/Ti*(r-y)+(v-u)/Tt

"Parameters
k:1
N:10
Ti:4
Tt:0.05
Td:1
b:0
dlow:-1e3
dhigh:1e3

END


CONNECTING SYSTEM conn

TIME t
r[reg]=1
y[reg]=y[proc]
v[reg]=u1
u1=if u[reg]<ulow then ulow else if u[reg]<uhigh then u[reg] else uhigh
u[proc]=u1
ulow:1
uhigh:1

END
```

```
MACRO fig33
"Simulation of the same system as in Fig31
"with regulator with anti-windup scheme

SYST  int pid conn
PAR N:0
PAR Ti:1
PAR b:1
PAR ulow:-0.1
PAR uhigh:0.1
SPLIT 1 1
AXES h 0 40 v 0 1.5
PLOT y[proc] r
STORE y[proc] u[proc] r i
PAR Tt:0.02
SIMU 0 40
SPLIT 3 1
AXES V 0 1.3
SHOW y r
AXES V -0.11 0.11
SHOW u
ASHOW i

END


MACRO fig34
"Simulation of system in Fig33 with different
"values of the reset time constant Tt

SYST  int pid conn
PAR N:0
PAR b:1
PAR Ti:1
PAR ulow:-0.1
PAR uhigh:0.1
AXES h 0 40 v 0 1.5
PLOT y[proc] r
STORE y[proc] u[proc] r
SIMU 0 40/c5
PAR Tt:1
SIMU/c4
PAR Tt:2
SIMU/c3
PAR Tt:3
SIMU/c2
PAR Tt:4
SIMU/c1
SPLIT 2 1
```

```
ASHOW y r/c1
SHOW y/c2
SHOW y/c3
SHOW y/c4
SHOW y/c5
AXES V -0.11 0.11
SHOW u/c1
SHOW u/c2
SHOW u/c3
SHOW u/c4
SHOW u/c5

END


MACRO fig52
"Simulation of jump and rate circuit

SYST jar conjar
AXES H 0 20 V 0 5
PLOT u[jar] y[jar]
PAR jlow:-0.5
PAR rlow:-0.5
STORE y[jar] u[jar]
SIMU 0 20

END


CONTINUOUS SYSTEM jar
"Jump and rate circuit

INPUT u
OUTPUT y
STATE x
DER   dx

e=u-x
y1=if e<jlow then jlow else if e<jhigh then e else jhigh
y2=if e<rlow then rlow else if e<rhigh then e else rhigh
dx=y2
y=y1+x

jlow:-1
jhigh:1
rlow:-1
rhigh:1

END
```

```
CONNECTING SYSTEM conjar

TIME t
u[jar]=if t<t1 then u1 else if t<t2 then u2 else if t<t3 then u3 else u4
t1:7
u1:4
t2:15
u2:1
t3:18
u3:3
u4:2

END

MACRO fig39
SYST select
STORE x1 x2 u v1 v2
SPLIT 1 1
AXES h 0 40 v -0.5 1.5
PLOT x1 x2 u
SIMU 0 40
SPLIT 2 1
ASHOW x1 x2
ASHOW u v1 v2
END

CONTINUOUS SYSTEM select
"Simulation of system with selector

TIME t
STATE x1  x2  i1  i2
DER  dx1 dx2 di1 di2

e1=r-x1
v1=k1*(b1*r-x1)+i1
e2=x2max-x2
v2=k2*e2+i2
v=min(v1,v2)
u=if v<ulow then ulow else if v<uhigh then v else uhigh
d=if t<t10 then 0 else d0

di1=k1*e1/Ti1+(u-v1)/Tt1
di2=k2*e2/Ti2+(u-v2)/Tt2
dx1=x2+d
dx2=u-x2

k1:1
b1:0.5
Ti1:2.5
```

```
Tt1=Ti1/5
k2:9
Ti2=9/25
Tt2=Ti2/5
x2max:0.2
ulow:-0.4
uhigh:0.4
r:1
d0:-0.15
t10:25

END
```