# Realization and tuning of PID controllers

**Author:**
*Jonas Wahlfrid, jonas_wahlfrid@hotmail.com*
*Student at the Programme in Software Engineering, Malmö University, Technology and Social Sciences*
*2007-02-19*

**Supervisor:**
*Ola Dahl*
*PhD in Automatic Control, Malmö University, Technology and Society*

## Abstract

This 5 point report discusses realisation and tuning of software based PID-controllers. The intended reader is an engineer that is interested in how a PID-controller is implemented and tuned with the AMIGO (Approximate M-constrained integral gain optimization) method. All mathematical reasoning is explained in detail step by step. The main purpose with this report is to gather knowledge from the reference sources, and clearly present this knowledge, so that it is accessible for engineers that work with PID-controllers.

- How does a PID-controller function?
- How can a PID-controller be implemented?
- How is a PID-controller tuned?

## Summary

This 5 point report deals with the realization and setup of software-based PID controllers. The intended reader is an engineer who is interested in how a PID controller is implemented and tuned using the AMIGO (Approximate M-constrained integral gain optimization) method. All mathematical reasoning is explained step-by-step and in detail. The main purpose of the report is to gather knowledge from the references, and formulate this knowledge so that it is easily accessible to engineers who practically work with PID controllers.
The aim of this report is to answer the following questions:

- How does a PID controller work?
- How can a PID controller be implemented?
- How is a PID controller trimmed using the AMIGO method?

# Table of contents

# 1 Introduction to Automatic Control

According to [1], control theory is "the study of how to use on-line measurements for automatic corrections in the process". The measurement value of the process is updated all the time, i.e. on-line. The controller constantly makes automatic corrections to the process in order to keep to the set point.

An example of regulation is cruise control for an electric car. The speed is measured with a sensor that senses the speed of the wheel, this is called feedback. The car's actual speed is compared with the desired speed and the car's computer calculates a signal that increases or decreases the throttle. Why does it have to be measured to measure, is it not enough to set the throttle to a certain position that corresponds to a certain speed? No, it is not enough because there are several disturbances that interfere with the process of converting chemical energy in the battery into kinetic energy. Examples of faults are changes in battery voltage due to charge level or age, downhill or uphill gradients, and changes in tire friction and bearing friction. Because the level of these disturbances varies, it is not possible to expect a certain position of the throttle to correspond to a certain speed. To compensate for the disturbances, the car's speed is feedback through on-line measurement, and the controller corrects the position of the throttle to comply with the setpoint.

According to [1], the idea of making machines that automatically vary one variable in order to maintain the level of another variable is an old one. Several devices from antiquity (700 BC-400 e.Kr.) were used to regulate water flow using a float, lever and a valve. An example of a mechanical proportional governor is James Watt's centrifugal governor for steam engines from the late 1700s. This regulator regulates the flow of steam by turning two balls driven by the main shaft of the steam engine. As the speed increases, the balls are lifted by centrifugal force, which causes a valve to cut off the flow of steam and the speed decreases.

# 2 On-off regulator

For simple controls, it is sometimes sufficient to have a regulator that only has two positions, on or off [1]. The most common example of an on-off regulator is the thermostat of an electric heating element. A thermostat is usually a mechanical switch that switches position with the temperature. On-off control is not an accurate control because the control signal adopts the maximum or minimum value. For example, it is not acceptable to implement cruise control with the help of an on-off regulator. Admittedly, the average speed might be kept around e.g. the setpoint of 50km/h, but the car's instantaneous speed would oscillate between, for example, 0 and 200 km/h, which is not acceptable. The control signal u assumes the maximum value if the control error $e > 0$ and the minimum value if $e < 0$. The control error e = setpoint $y_{sp}$ – actual value y. Usually hysteresis or a dead zone is used to prevent the frequency of switch-on/switch-off from becoming too great [5, p. 4].

# 3 P, PI and PID controller

A more accurate controller than the on-off controller is the PID controller because the control signal can assume continuous values [1]. PID is an abbreviation for proportional, integrative and derivative action. Typically, these different parts are combined into PI or PID controllers [1]. The job of a controller is to keep the actual value close to the set point by changing the control signal. In the cruise control example in Figure 3.1, the actual value is the car's speed in km/h, the setpoint is the desired speed in km/h and the control signal varies between 0 Volt and 10 Volt. The control signal corresponds to 0-100 Volts to the DC motor that powers the electric car. The control error or deviation is the difference between the setpoint and the actual. Table 3.1 lists the denominations of variables as [5].

| Variable | Description |
|---|---|
| y | actual value, measured value, process value |
| u | Control signal |
| YSP | Setpoint, SP is an abbreviation for "set point" |
| e | Control error, control deviation E = YSP – y |

**Table 3.1** Table of Variables [1].

To show an abstraction of a control circuit, a block diagram is used. An example of the cruise control block diagram is shown in Figure 3.1.

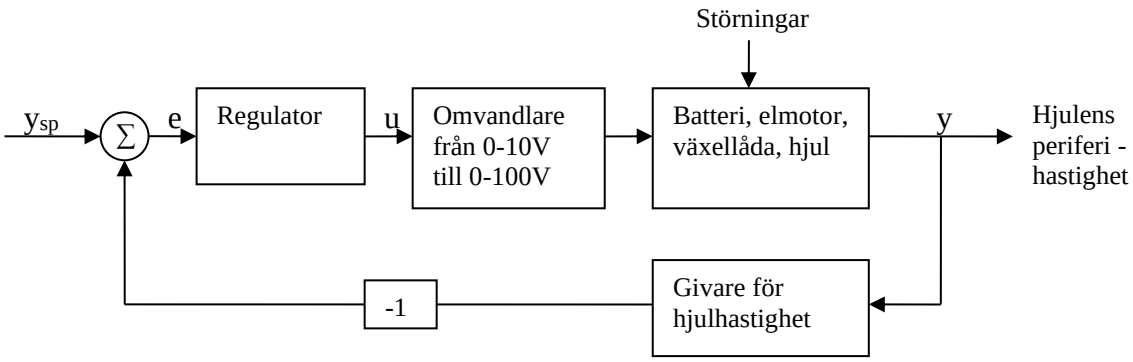


**Figure 3.1** Block diagram showing the cruise control schematically, principle from [3, p. 10][5, p. 3].

## *3.1 Operation of the P-controller*

Suppose we try to use a P (proportional regulator) to build a cruise control. The following equation shows an example of the dependence between the voltage from the regulator and the measured velocity y. The control signal cannot assume values higher than 10 Volts and not lower than 0 Volts.

$$u = 0{,}1(90 - y) + 9 \text{ V}$$

90km/h is the desired speed and 9 V is the voltage required to maintain 90km/h on a flat road and normal air resistance. The factor of 0.1 is called amplification and indicates how strongly the regulator should act on a deviation. If the speed decreases to 85km/h due to e.g. an uphill slope, the controller will increase the voltage to

$$0{,}1(90 - 85) + 9 = 9{,}5 \text{ V}$$

The formula for the P regulator is [1]

$$u = K(y_{sp}-y) + u_b$$

Since e = $y_{sp}$ - y, the formula can be rewritten to [1]

$$u = K\,e + u_b$$

Where K is the regulator gain and $u_b$ is an offset for the control signal to maintain a certain level when the control error is zero. The gain K indicates how much the control signal should react to a change in the control fault. If a process has a low gain, a higher value of K is chosen than if the process has a high gain, then a low value of K is chosen. K is thus chosen inversely proportional to the strengthening of the process. If the process gain is negative, which it is in the case of e.g. cooling, a negative value of K is entered. K is thus set to a positive value if the measured value goes in the same direction as the control signal, otherwise negative.

Unfortunately, the P-controller does not work very well because it gives a residual control error unless the offset changes with each setpoint change. Suppose we change the setpoint to 50km/h without changing the offset

$$0{,}1(50-85)+9 = 5{,}5\ \text{V}$$

The control signal decreases to 5.5 V in the event of a control fault of –35 km/h. However, in the event of a control error on the
–15km/h increases the control signal/voltage in relation to a control fault of –35km/h, according to

$$0{,}1(50-65)+9 = 7{,}5\ \text{V}$$

As the control error decreases, the control signal increases and thus the speed. In the case of control error 0, a control signal/voltage of 9 V is obtained, which corresponds to about 90 km/h, which is far from the setpoint of 50 km/h. This means that it will be impossible to arrive at the setpoint.

In practice, the actual value will stabilise at e.g. 70 km/h, i.e. a control error of 50-70 = -20 km/h, because the force driving the car forward is in equilibrium with the air resistance. In other words, it is impossible to reduce the control error to 0. If the gain is increased a lot, it is possible to get close to zero in control errors, but high gain will lead to instability in the form of self-oscillation. Although the setpoint and the offset are always the same, a control deviation is obtained if a disturbance in the form of, for example, an uphill slope occurs.

The P-controller is not capable of maintaining the setpoint (setpoint tracking) or fault suppression without permanent control failures, unless the value of the offset changes according to the circumstances [1]. Changing the offset according to setpoint and circumstances manually is not practical, making the P-controller almost unusable [1]. An example of a mechanical P-regulator is the thermostatic valve on a water element in a building.

## *3.2 PI controller function*

The PI controller has a proportional and an integrating part. The integral part replaces the offset constant in the P controller so that the adjustment is automatic. The integral part is proportional to the integral of the control error [1].

### 3.2.1 What is meant by an integral?

Figure 3.2 shows how a car's speed changes over time.

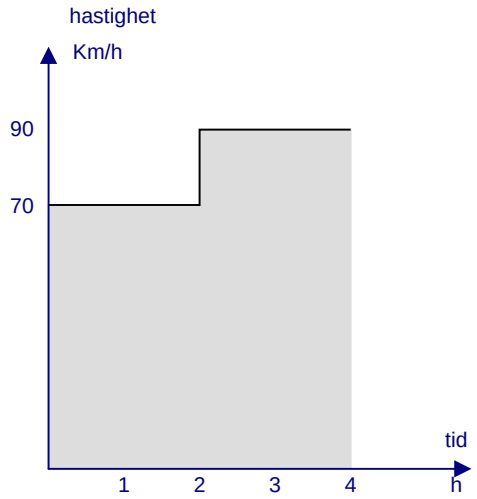


**Figure 3.2** Speed of a car over four hours.

How far does the car in Figure 3.2 travel during the first two hours?
Distance = Speed * Time
Distance = 70 * 2 = 140km

The area of the shaded area in Figure 3.2 between 0 and two hours is Area = 70 * 2 = 140

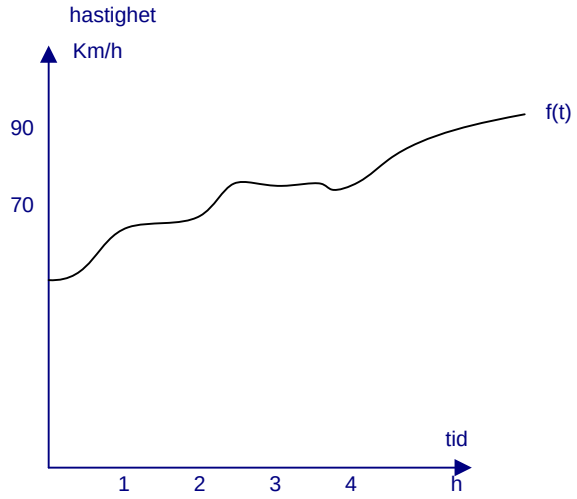


**Figure 3.3** A car's speed over four hours.

The area between the curve and the timeline in Figure 3.3 is a measure of how far the car has travelled. The speed changes according to the function f(t) in Figure 3.3.

If the distance, i.e. the area, is desired between time 1 and 4, this can be indicated by the symbol

$$\int_1^4 f(\tau)d\tau$$

and the integral of f(x) is read from 1 to 4 [4].

### 3.2.2 PI controller equation

The general formula for the PI controller is [1][2][3][5]

$$u(t) = K\left( e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau \right)$$

If K is multiplied into the parentheses, the formula can be rewritten to [1]

$$u(t) = K\,e(t) + K\frac{1}{T_i}\int_0^t e(\tau)d\tau$$

This formula reads: The controller control signal u at time t is the sum of two parts. The first is the P part, the gain K multiplied by the control error at time t. The second part is the integral part, the amplification times the inverse of integral time Ti multiplied by the integral of the control error from time 0 to time t. If the integration time Ti is large, the adjustment will be slow because the integral is multiplied by 1/Ti. The quantity of Ti is time and is usually given in seconds or minutes [1].

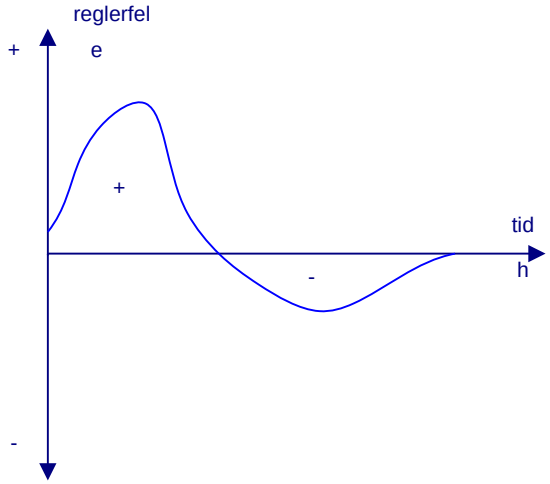


**Figure 3.4** The integral of the control error e with positive and negative contribution [2].

If the control error is positive, the integral of e(t) increases. If the control error is negative, the integral of e(t) decreases. The integral part is a weighted sum of all old control errors [2], see Figure 3.4. Figure 3.5 shows the result of a setpoint change if we use a PI controller to regulate, for example, a car's speed.

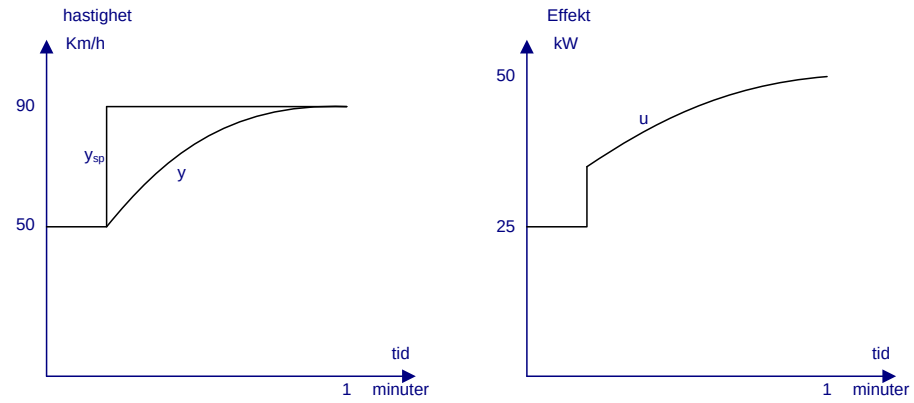


**Figure 3.5** Diagram showing setpoint change from 50km/h to 90km/h.

With a PI controller, there is no permanent control error because the integral of e(t) decreases or increases as long as the control error is not zero. The integral part affects the control signal, which means that the actual value is closer to the set point. This applies not only to setpoint changes, but also to disturbances.

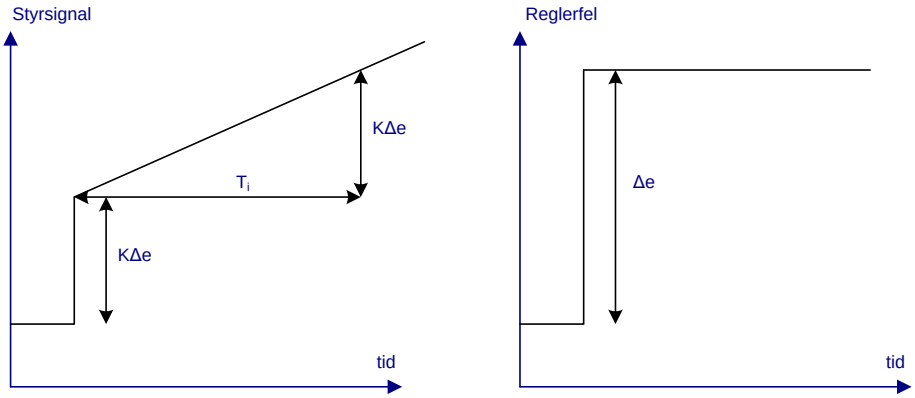


**Figure 3.6** Graphical interpretation of the integration period Ti [2].

The change in the control signal in the event of a change in the control fault is given in accordance with [2] by

$$\Delta u(t) = K(1 + 1/T_i)\Delta e\,.$$

The symbol Δ is the Greek letter delta, which in this case denotes difference, i.e. Δe is the difference of the control error from the past value to the present value. The integration period can be illustrated as shown in Figure 3.6 as shown in [2]. The control signal is first changed with the contribution of the P part, i.e. K Δe. Subsequently, the control signal grows linearly at a speed determined by Ti. The control signal grows to 2 K Δe after the time t = Ti, which means that the P stage has doubled. The integration time can be interpreted as the time it takes to double the P stage when the control error is constant. If the process is fast then Ti is selected short, while a slow process requires longer $T_{in}$ time because the control signal should change more slowly. In other words, the integration time is chosen proportionally to the time of the process [2].

### 3.2.2.1 Limitation of the control signal

The control signal has a lower limit and an upper limit. In order to prevent the integral of e(t) from growing even though the control signal has reached a limit, the integral part must be limited if the control signal reaches a limit. If the integral part is allowed to grow despite the fact that the control signal has reached the maximum or the minimum limit, so-called integrator winding is achieved, which must be avoided. The function that limits the integral part is called anti-windup. If the anti-windup is not used, the integral part grows positively or negatively, even though the control signal cannot influence the process, because the control signal has reached the maximum or minimum value. It is not desirable for the integral part to grow uncontrollably because it then takes a long time to reduce the integral part again when the control signal turns, which can lead to poor stability. One anti-windup method is to stop updating the integral part if you reach max or min, another is to let the difference between the limited control signal and the unlimited control signal affect the value of the integral. More information on anti-windup methods is given in section 4.

### 3.2.2.2 Starting value of the I-part

In order to reduce the time of the oscillation process after starting a regulator with integral action, the integral part can be given a starting value [3, 56].

## 3.2.3 Setpoint weighting factor b

According to [1], a controller has two main functions, setpoint tracking and fault suppression. To prevent a regulator from reacting so violently to a setpoint change, setpoint weighting can be used. The formula for a PI controller with setpoint weighting is given below [1][5].

$$u(t) = K\left( b\, y_{sp}(t) - y(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau \right)$$

If K is multiplied into the parentheses, the formula can be rewritten to

$$u(t) = K\left(b\, y_{sp}(t) - y(t)\right) + K\frac{1}{T_i}\int_0^t e(\tau)d\tau$$

The setpoint weighting factor (b) is given values between 0 and 1. Figure 3.7 shows that the control signal does not react as strongly to a setpoint change of b < 1. The regulator's ability to suppress interference is not affected by the value of b.

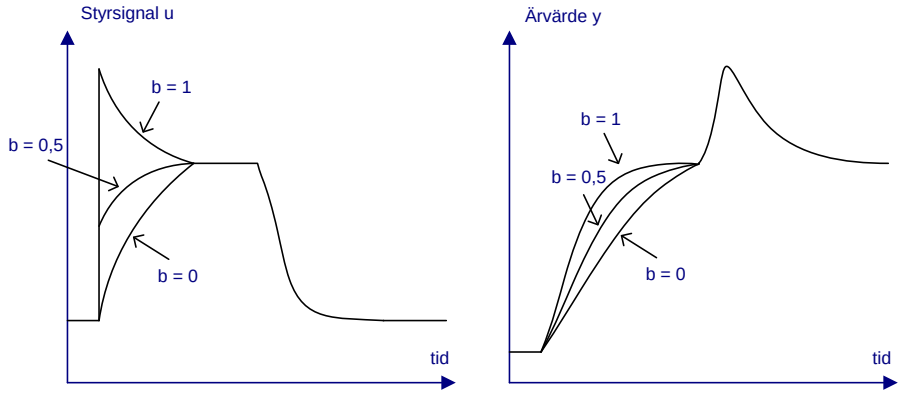


**Figure 3.7** PI controller, setpoint stage and step-shaped disturbance with different b values [1][5].

## *3.3 Function of the PID controller*

A characteristic of [2] that limits the PI controller is that it does not try to predict what will happen to the controller failure in the near future. The D part of a PID controller is proportional to the derivative of the control error. The D stands for derivative effect [1][2]. The idea of using the derivative of the control error is that the controller then has a predictive ability to estimate where the control error is heading. The different parts of the PID controller can be summarized as the P part represents the present, the I part the past tense, and the D part the future [1][5].

### 3.3.1 What is meant by derivative?

#### 3.3.1.1 Change rate

Figure 3.8 shows how a car moves from the starting position over time.

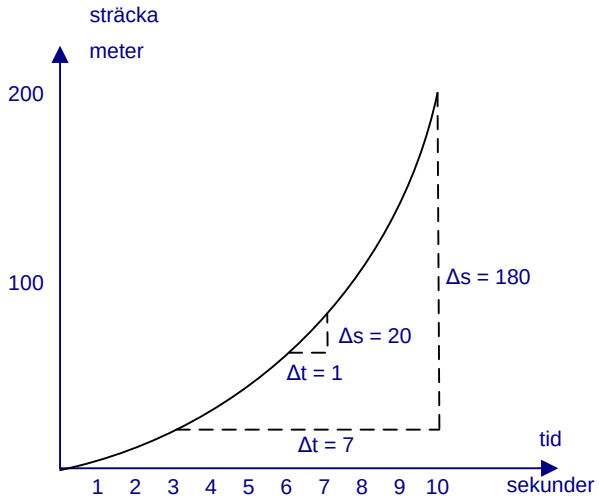


**Figure 3.8** A car's movement over 10 seconds.

What is the car's average speed between time 0 and time 10?
Distance s = Speed v * Time t
v = s/t
v = 200 / 10 = 20 meters/second

What is the average speed of the car during time 3 to time 10?
v = 180 / 7 = 25.7 m/s

What is the average speed of the car during time 6 to 7?
v = 20 / 1 = 20 m/s

What is the speed of the car at time 6?
If Δt decreases or, in other words, goes towards zero, then the instantaneous velocity is given to 20 m/s. The speed of 20 m/s is a so-called limit value. In other words, the velocity at a given moment is the limit that the rate of change approaches when the time interval approaches zero. This is written $\Delta t \rightarrow 0$.
Velocity = limit value of $\Delta s / \Delta t$ then $\Delta t \rightarrow 0$.

### 3.3.1.2 Slope of a curve

The coefficient of direction k of a straight line is determined by $k = \Delta y / \Delta x$. Figure 3.9 draws a straight line through points (6,60) and (6,5 , 70).

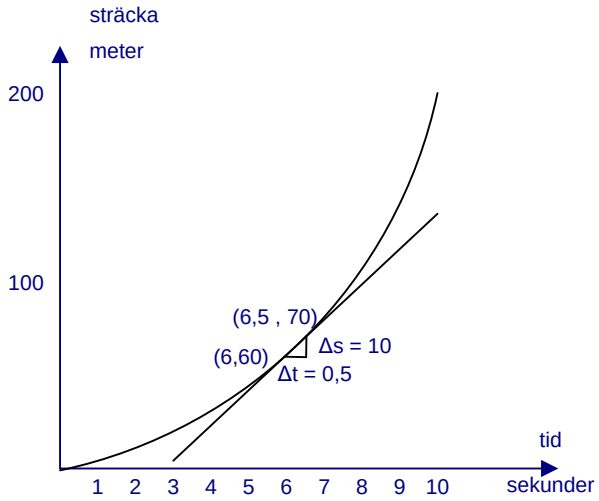


**Figure 3.9** A car's movement over 10 seconds and a straight line drawn.

The directional coefficient of the straight line is $k = \Delta s / \Delta t = 10 / 0.5 = 20$ m/s. In other words, the coefficient of direction of the line corresponds to the average speed between the points.
Assume that the point (6.5, 70) is a moving point on the curve. As (6.5, 70) approaches the point (6.60), Δt becomes smaller and smaller. Velocity = limit value of $\Delta s / \Delta t$ then $\Delta t \rightarrow 0$. The line through the point (6.60) and (6.5 , 70) will approach a limit position where it touches the curve of the point (6.60). The line is then one tangent to the curve at the point (6.60). The derivative = the k-value of the tangent [4].

### 3.3.1.3 Definition of the derivative

How is the k-value/derivative of a tangent to an arbitrary function y = f(x) determined?

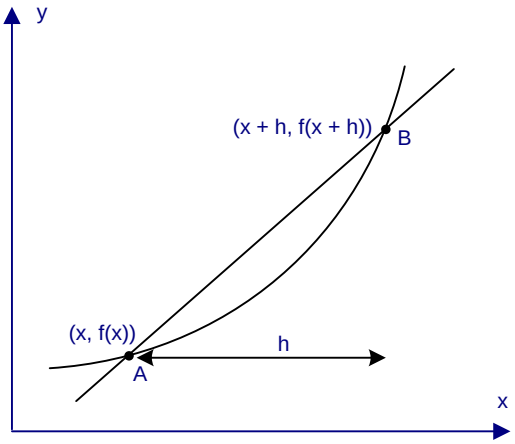


**Figure 3.10** Straight line through points A and B [4].

The line passing through points A and B in Figure 3.9 has the k-value

$$k = \frac{\Delta y}{\Delta x} = \frac{f(x+h) - f(x)}{x + h - x} = \frac{f(x+h) - f(x)}{h}$$

Now let's move point B closer to point A. This means that the distance h becomes smaller and smaller. This is called h going to zero, h → 0.

The tangent at point A has the k value

$$k = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

This is called the derivative's definition [4]. Limes is abbreviated lim which means border in Latin.

The derivative = the k-value of the tangent [4].

The derivative of a function f(x) indicates how quickly it changes as the independent variable x increases. The derivative is usually denoted $f'(x)$ (pronounced 'f-prim of x') or df/dx (pronounced 'd-f, d-x').
f(x) = x2 has the derivative $f'(x) = 2x$
f(x) = x2 has the derivative df/dx = 2x (other denomination)

### 3.3.2 PID controller equation

The general formula for a PID controller is [1]

$$u(t) = K\left( b\, y_{sp}(t) - y(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\, e'(t) \right)$$

Where $T_d$ is the derivative time. If K is multiplied into the parentheses, the formula can be rewritten to

$$u(t) = K\left(b\, y_{sp}(t) - y(t)\right) + K\,\frac{1}{T_i}\int_0^t e(\tau)d\tau + K\, T_d\, e'(t)$$

This formula reads: The controller control signal u at time t is the sum of three parts. The first is the P part, the setpoint ysp multiplied by the setpoint weighting factor b, minus the actual value y at time t, which is multiplied by the gain K. The second part is the integral part, the amplification times the inverse of integral time Ti multiplied by the integral of the control error from time 0 to time t. If the integration time Ti is large, the adjustment will be slow because the integral is multiplied by 1/Ti. The quantity of Ti is time and is usually given in seconds or minutes [1]. The third part is the derivative part, the gain multiplied by the derivative time Td times the derivative of the control error at time t. If the derivative time Td is large, an estimate of the control error is obtained for a longer period of time in the near future. The quantity of Td is time and is usually given in seconds or minutes. An example of a cruise control block diagram is shown in Figure 3.11.

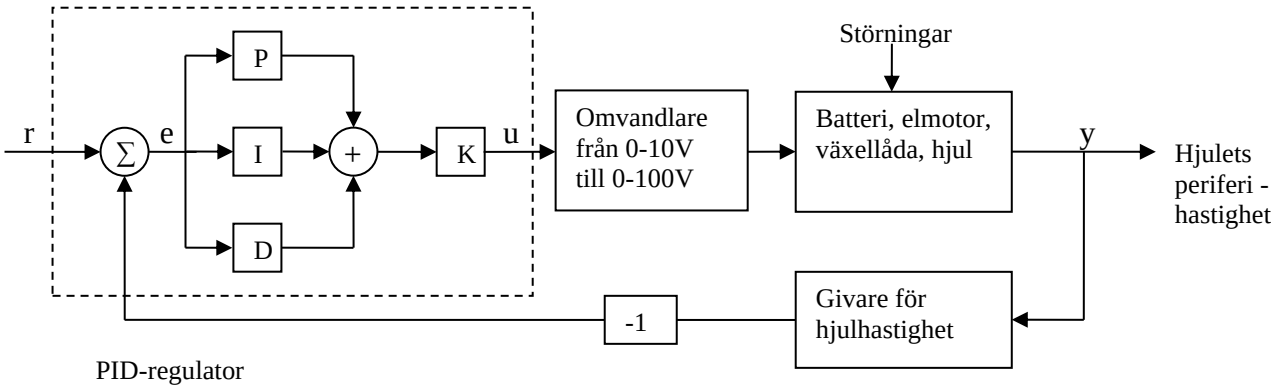


**Figure 3.11** An example of a cruise control block scheme, realized with a PID controller. The output of a PID controller is the sum of three parts. Principle from [1][3][3, p. 10][5, p. 3].

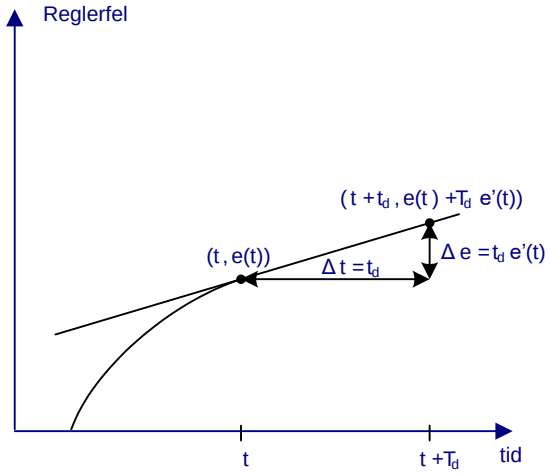


**Figure 3.12** Interpretation of the derivative time Td. Principles from [2] and [4].

Figure 3.12 shows that the derivative nobility of the regulator makes it possible to predict the control error time Td in the future. A straight line is used to extrapolate the function e(t) up to time t + $T_d$. The coefficient of direction k of a straight line is determined by k = Δy / Δx. The straight line is a tangent to the function e(t). The derivative of e(t) = e′(t) which is the same as the k-value of the tangent.

$$\Delta e = T_d e'(t)$$

$$e(t + T_d) = e(t) + T_d e'(t)$$

For the prediction to be useful, the derivative time Td cannot be too large, because the prediction is only reliable for a limited time into the near future. The reason for this is that the reliability of a prediction deteriorates if the time span into the future is increased. In fast processes, the control error changes rapidly, which means that Td must be chosen to a smaller number than if the process is slow. The derivative time shall be chosen proportional to the times of the process [2].

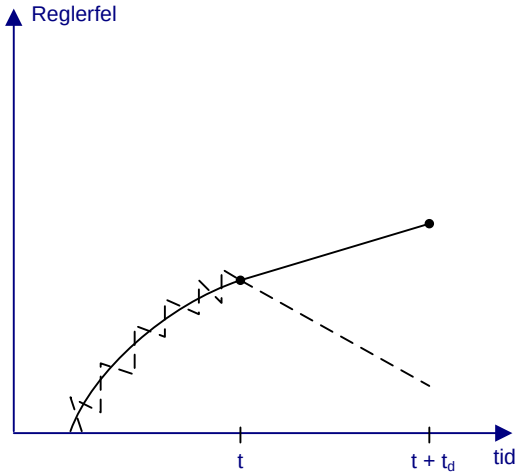


Figure 3.13: The dashed line shows a signal with noise that makes the prediction td units of time into the future unreliable. The solid line indicates a signal that does not contain noise.

If the measurement signal y contains noise, it is inappropriate to use the derivative part because the prediction is not reliable, see Figure 3.13. The derivative part amplifies the noise if a noisy signal is not filtered. If a noisy measurement signal is filtered through a low-pass filter that attenuates high frequencies, the functionality of the derivative adement can be improved. The disadvantage of filtering according to

[1, p. 60] is that the filter gives rise to a certain delay/phase shift of the signal. A delay reduces the reliability of the derivative adement's prediction, and can also impair the stability of the control system. It is important to find a balance between how hard you want to filter the measurement value and how much delay you can allow [1, p. 61].

If the setpoint changes, then e changes by the same amount, which means that the derivative part becomes very large. The explanation for this is that, if the magnitude of e at time t is very different from e at time t + $T_d$, then the straight line in Figure 3.12 becomes steep. To avoid this, let the D part act on the measured value y instead of the control error
e = setpoint ysp — actual value y. This gives the following equation where yf is the filtered measurement signal [1, p. 58]

$$u(t) = K\left( b\, y_{sp}(t) - y(t) + \frac{1}{Ti}\int_0^t e(\tau)d\tau - T_d y_f'(t) \right)$$

which can also be written

$$u(t) = K\left(b\, y_{sp}(t) - y(t)\right) + K\frac{1}{T_i}\int_0^t e(\tau)d\tau - K\, T_d y_f'(t)$$

# 4 Implementation of a PID controller in software

## *4.1 Components of discrete-time control systems*

A PID controller can be implemented in several ways, e.g. with the help of analogue technology or software in a computer. A controller implemented in software works in discrete time while the physical system that is regulated works in continuous time. Discrete time means that the control signal only changes at discrete times, e.g. once per second or more frequently [3]. The measurement signal is also discreet because it is sampled at specific times.
To build a discrete-time control system, the components in Figure 4.1 [3, p. 263] are needed.

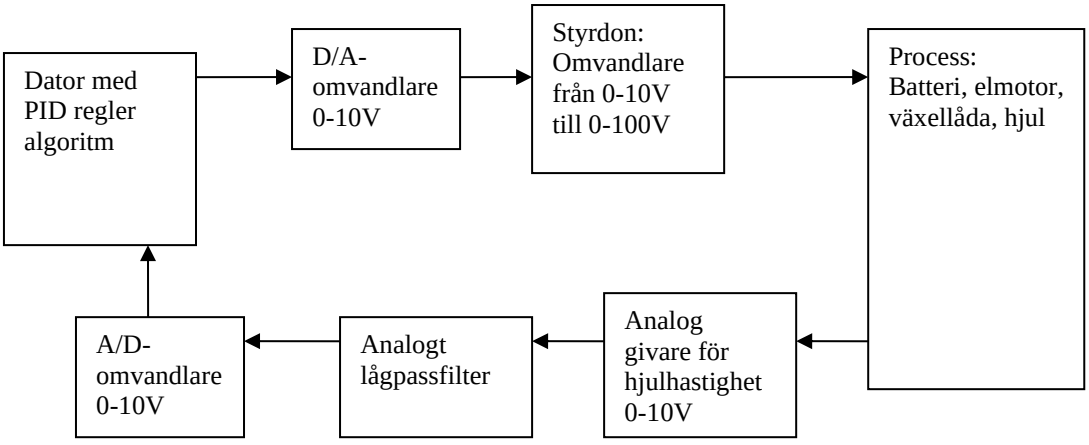


**Figure 4.1** Computer-based cruise control for a car, principle from [3, p. 264].

The different elements in Figure 4.1 are described below.

**Sensor:** The sensor measures the actual value of the process, which in this case is the wheel speed.

**Analogue low-pass filter:** The alias effect means that high frequencies are misinterpreted as low frequencies due to the measurement being sampled, see Figure 4.3. To avoid the alias effect, there must be a low-pass filter before the A/D converter. A low-pass filter allows signals with frequencies lower than the latching frequency to pass through, while higher frequencies are attenuated, see Figure 4.2. The filter must eliminate any frequencies higher than half the sampling rate [5, p. 414]. Unless a low-pass filter is used, high frequencies will be misinterpreted as low frequencies, which means that the controller will regulate on incorrect readings. An example of when frequencies higher than half the sampling rate can occur is the measurement of the level in a tank whose contents are sloshing.

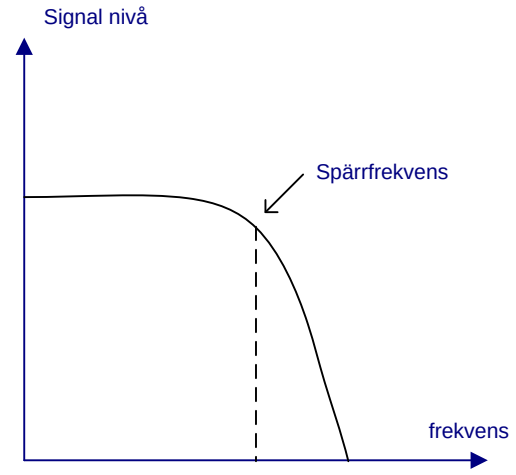


**Figure 4.2** Low-pass filter [6].

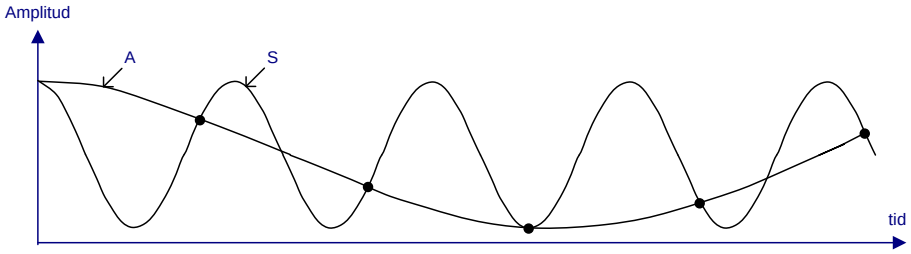


**Figure 4.3** The signal S is perceived after sampling as the alias signal A [5, p. 414].

A simple low-pass filter can be created with a resistor R and a capacitor C [6, p. 45], see Figure 4.4.

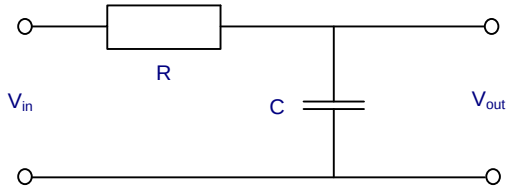


**Figure 4.4** First-order low-pass RC filters [6, p. 45].

A capacitor can store electrical energy in an electric field. The current I through capacitor C is equal to the derivative of the voltage $V_{out}$ across the capacitor multiplied by the capacitance C measured in coulomb/volts or Farad.
$I = C\, V'_{out}(t)$

The voltage $V_{out}$ across the capacitor is given by
$V'_{out}(t) = \frac{I}{C}$

If the frequency across the capacitor in Figure 4.4 is increased, the current through the capacitor increases, causing the voltage drop across the resistor to increase. An increased voltage drop across the resistor in turn leads to Vout being lower than Vin. It is possible to describe how Vout depends on Vin with the help of Kirchhoff's law of tension. Kirchhoff's law of voltage (also called Kirchhoff's 2nd law) applies to voltages in an electrical network and reads as follows: The sum of all branch voltages included in a loop is zero. Ohm's law: U = R I. Kirchhoff's law of tension applied to Figure 4.4 gives
$V_{in} - V_r - V_{out} = 0$

where $V_r$ is the voltage drop across the resistor. By using Ohm's law, Vr can be replaced by R I, which gives
$V_{in} - R\,I - V_{out} = 0$

Which can be rewritten as
$V_{in} - V_{out} = R\,I$
$R\,I = -V_{out} + V_{in}$

I can be replaced by $C\, V'_{out}(t)$ which gives the following differential equation
$R\,C\, V'_{out}(t) = -V_{out}(t) + V_{in}(t)$

Since the unit of R is volts/amps and the unit of C is amperes/(volts/s),
$T_f = R\,C$ the unit has seconds s.

The low-pass filter in Figure 4.4 then has the following equation
$T_f\, V_{out}'(t) = -V_{out}(t) + V_{in}(t)$

The relationship between a low-pass filter's time constant Tf and its cut-off frequency is
Tf = 1 / the locking frequency.

**A/D converters:** The analog to digital converter's job is to convert the analog signal into a binary signal that can be read by the computer. The resolution of the converter is dependent on how many bits are used to represent the measured value.

**Computer:** A PID control algorithm is executed in the computer that reads the value from the A/D converter and calculates a value of the control signal. The value of the control signal is calculated as a binary number to the D/A converter.

**D/A converter:** The digital to analog converter is used to convert the binary signal into an analog signal.

**Actuator:** The task of the ECU is to influence the controlled process variable. In Figure 4.1, it is the voltage that is changed to make the electric motor rotate at a certain speed so that the wheel speed can be affected.

## *4.2 Discrete PID controller*

A time-continuous PID controller consists of three parts P, I, and D part according to the equation $u(t) = K(b\, y_{sp}(t) - y(t)) + K\frac{1}{T_i}\int_0^t e(\tau)d\tau - K\, T_d\, y_f'(t)$

A list of the variables that will be used in this chapter is listed in Table 4.1.

| Variable | Description |
|---|---|
| u | Control signal (limited control signal) |
| v | Unlimited control signal |
| ulow | Minimum value of u |
| uhigh | Maximum value of u |
| K | Amplification K |
| b | Setpoint weighting factor, value 0-1 |
| YSP | Setpoint, sp is an abbreviation for "set point" |
| y | Actual value |
| Ti | Integration time in e.g. seconds |
| Tt | Time constant in e.g. seconds that controls how quickly the integral part is to be reset after a limitation of the control signal. $T_t$ is referred to as "tracking time constant" in English. |
| e | Control error, control deviation e = ysp – y |
| Td | The derivative time in e.g. seconds |
| YF | Filtered actual value |
| h | Sampling time in e.g. seconds |
| P | The proportional part |
| In | The integral part |
| Tk | The term tk is used instead of absolute time t. Each sample is numbered consecutively, where k denotes the sequential number.<br>The following applies to tk - tk-1 = h |
| t | Time in e.g. seconds |
| D | The Derivative Part |
| Acting | The time constant of the low-pass filter in e.g. seconds |
| yfold | Same as yf(tk-1), the filtered metric from the previous execution |
| Df | Filtered Derivative Nobility |

**Table 4.1** Variable Table [1].

### 4.2.1 P-section

The time-continuous proportional part is given by:

$$P(t) = K(b\, y_{sp}(t) - y(t))$$

The proportional part can be implemented directly in discrete-time form by replacing the continuous variables with sampled variables [5, p. 415].

$$P(t_k) = K(b\, y_{sp}(t_k) - y(t_k)) \quad (4.1)$$

### 4.2.2 Part I

The continuous integral part is given by [5, p. 415]

$$I(t) = K\frac{1}{T_i}\int_0^t e(\tau)d\tau$$

This equation can be approximated by the sum of all rectangle areas obtained if each control error value e(tk) up to the last sample tk is multiplied by the length of the sampling time h. The shorter the sampling time h, the better the approximation [3, p. 269]. See Figure 4.5.

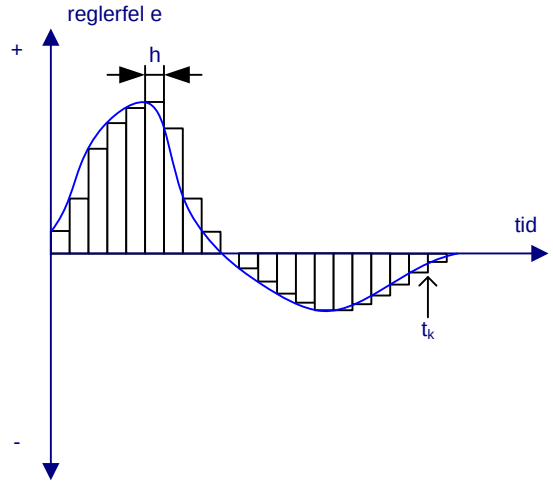


**Figure 4.5** Principle of approximation of the integral of the control error [3].

If the area function with characters in Figure 4.5 is denoted by A(t), then the area of

$$A(t) = \int_0^t e(\tau)d\tau$$

The area function A(t) is a primitive function of e(t), which means that

$$A'(t) = e(t)$$

According to the definition of the derivative, the derivative can be approximated by the following forward difference

$$A'(t) \approx \frac{e(t+h)-e(t)}{h}$$

The continuous integral part is given by [5, p. 415]

$$I(t) = K\frac{1}{T_i}\int_0^t e(\tau)d\tau$$

A derivation of the above equation gives [5, p. 415]

$$I'(t) = K\frac{1}{T_i}e(t)$$

This can be rewritten using the derivative's approximation to the below. The term tk is used above instead of absolute time t. Each sample is numbered consecutively, where k denotes the serial number, the following is true when $t_k$ - tk-1 = h.

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

The equation above can be written as a recursive equation [5, p. 415]

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

Which can be interpreted to mean that, h e(tk) gives a rectangle area that is multiplied by K and 1/Ti. I(tk) contains the results of previous calculations.

The discrete-time form can be written as [5, p. 415]

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

### 4.2.3 The I-part with anti-windup

The control signal u has a minimum value and a maximum value because a control unit such as the converter in Figure 4.1 cannot increase the voltage to more than 100 volts, nor can the voltage be lowered to less than 0 volts. It is not desirable to allow the integral part to grow positively or negatively even though the control signal has been restricted. The reason for this is that when the control error later changes characters, it takes a long time for the integral part to change so that the control signal changes. For example, in order to keep to the speed setpoint in the example from Figure 4.1, the control signal cannot be allowed to stick to the minimum or maximum value, pending the reset of the integral part. The integral part must be limited so that the integral part does not grow if the minimum or maximum control signal is reached, this is called integrator winding. In order to prevent wind-up, an anti-wind-up mechanism must be introduced.

An anti-windup method is to use "Back-calculation and tracking", this method is described below [5, p. 80]. An I-regulator without anti-windup is given by

$$I(t) = K\frac{1}{Ti}\int_0^t e(\tau)d\tau$$

The following function saturateOutput is introduced as an auxiliary function to limit the unlimited control signal so that the limited control signal u does not assume values outside the minimum ulow or maximum uhigh [5, p. 428].

$$u(t_k) = saturateOutput(v(t_k), uLow, uHigh) \quad (4.2)$$

Given the parameters v, uLow, and uHigh, the saturateOutput function returns a finite value u as shown below.

If uLow ≤ v ≤ uHigh is returned u = v

If v > uHigh is returned, u = uHigh

If v < uLow, u = uLow is returned

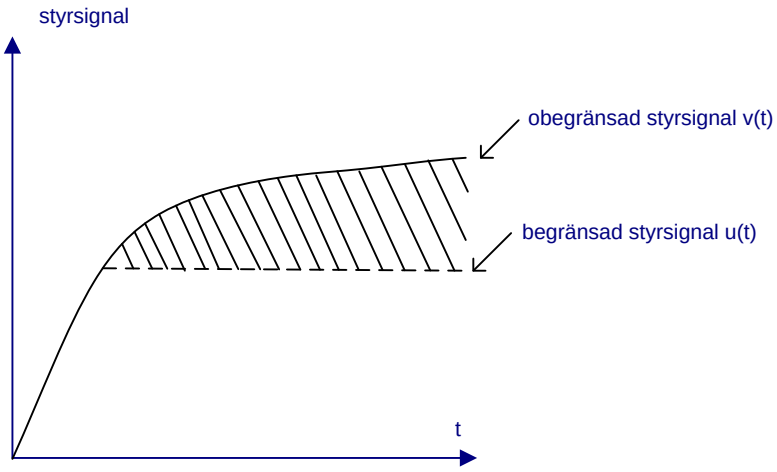


**Figure 4.6** Unlimited control signal v(t) and limited control signal u(t).

The negative area of the dashed area in Figure 4.6 is given by:

$$A_{streck} = \int_0^t [u(\tau) - v(\tau)]\, d\tau$$

This is the area of characters that is added to the I part so that the I part does not exceed or fall below the ulow and uhigh constraints. It would be conceivable to directly adjust the I-part with the entire dotted area in Figure 4.5, but this is not desirable, since a temporary change in the measurement signal can cause the derivative part and the proportional part to contribute a value that saturates u, and thus inadvertently resets the integral part.
To avoid this, the time constant Tt. The time constant Tt is called "tracking time constant" [5, p. 80].

$$A_{streck} = \frac{1}{T_t} \int_0^t [u(\tau) - v(\tau)]\, dt$$

The time constant $T_t$ determines how quickly the integral part is to be restored after limiting the control signal. If the time constant Tt is chosen too small, a temporary change in the measurement signal can cause the derivative part and the proportional part to contribute a value that saturates u, and thus inadvertently resets the integral part. An example of when this can happen is when an agitator in a tank is started, causing waves to form in the tank that affect the measurement signal.
The time constant Tt should be greater than Td but less than Ti. A rule of thumb is to choose $T_{t\ according}$ to $T_t = \sqrt{T_i\, T_d}$ [5, p. 80] for a PID controller and for a PI controller $Tt = 0{,}5\ T_{in}$ [7, p. 5].

The continuous-integral part of the anti-windup is given by the following equation, principle from [6, p. 289]

$$I(t) = K \frac{1}{T_i} \int_0^t e(\tau) d\tau + \frac{1}{T_t} \int_0^t [u(\tau) - v(\tau)]\, d\tau$$

The above equation can be approximated according to the same principle as in Section 4.2.2. After a derivation of the above equation, the

$$I'(t) = K \frac{1}{T_i} e(t) + \frac{1}{T_t} (u(t) - v(t))$$

This can be rewritten using the derivative's approximation to

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

Which can be written as a recursive equation

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

Which can be interpreted to mean that, h e(tk) gives a rectangle area that is multiplied by K and 1/Ti. I(tk) contains the results of previous calculations. h (u(tk)-v(tk)) gives a rectangle area for the part that exceeds the limited signal u(tk) multiplied by 1/Tt.

The discrete-time equation above can be rewritten, replacing the control deviation e with $y_{sp} - y$ to

$$I(t_{k+1}) = I(t_k)+\frac{K\,h}{T_i}(y_{sp}-y)+\frac{h}{T_t}(u(t_k)-v(t_k))$$

To save CPU power, the equation above can be rewritten to form three parts. Parameter equations 4.3 and 4.4 are executed only if any variable in the equations is changed [5, p. 428, F13.19].

$$p3 = \frac{K\,h}{T_i} \quad (4.3)$$

$$p4 = \frac{h}{T_t} \quad (4.4)$$

$$I(t_{k+1}) = I(t_k)+p3(y_{sp}-y)+p4(u(t_k)-v(t_k)) \quad (4.5)$$

### 4.2.3.1 Conditional integration

The risk with the anti-windup function in section 4.2.3 is that a temporary change of the measurement signal y can cause the D-part to generate a large value, which resets the I-part if the time Tt is reversed. According to [9], there is a method called conditional integration that does not have this risk. This method is a simple and easy to explain. According to [9], conditional integration is recommended in most contexts, see Figure 4.7.

```
while(true) {
   y=getProcessVariable() // get process variable
   e=ysp-y                 compute error
   P=K*e                   proportional party
   D=K*Td*(e-eold)/h       derivative party
   v=P+I+D                 compute nominal output
   u=saturateOutput(v, uLow, uHigh) // saturate output
   setManipulatedVariable(u)  set manipulated variable
   if u==v {
      I=I+K*h/Ti*e         integral part
   }
   eold=e                  update e old
   wait(h)                 wait h seconds
}
```

**Figure 4.7** Conditionally integrated regulator. Differentiation on e and no filter on the D-part.

### 4.2.4 Part D

The continuous derivative part is given by

$$D(t) = -K\, T_d\, y_f'(t)$$

An approximation of the derivative in the equation above is obtained by using the backward difference as shown below.

$$y_f'(t_k) \approx \frac{y_f(t_k) - y_f(t_k - 1)}{h}$$

The derivative of the filtered actual value yf at time tk is approximated by calculating the difference between the current sampling $y_f(tk)$ and the previous sampling $yf(t_{k-1})$ divided by the sampling interval h. The shorter the sampling time h, the better the approximation [3, p. 268].

The discrete-time form of the continuous-time equation above can be written according to [3, p. 269] as

$$D(t_k) = -K\, T_d \frac{y_f(t_k) - y_f(t_{k-1})}{h}$$

The sampling points and the predicted actual value are illustrated in Figure 4.8.

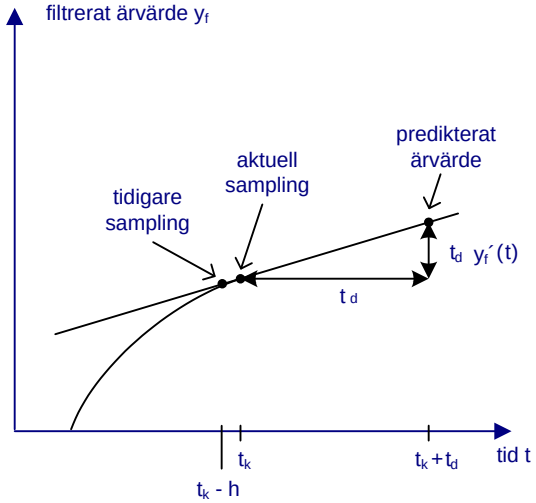


**Figure 4.8** Illustration of sampling points and the predicted actual value.

To save CPU power, the equation above can be rewritten to form two parts. Parameter equation 4.6 is executed only if any variable in the equation is changed, partly from [5, p. 428, F13.19].

$$p5 = -K \frac{T_d}{h} \quad (4.6)$$

$$D(t_k) = p5\left(y_f(t_k) - y_f(t_{k-1})\right) \quad (4.7)$$

### 4.2.4.1 Digital low-pass filter

As mentioned earlier in section 3.3.2, the measured value must be filtered in order for the prediction of the D-part to be reliable if the test signal contains noise, see also Figure 3.13. A D-part without a filter should not be implemented because the amplification of the noise will be too great. The disadvantage of filtering according to [1, p. 60] is that the filter causes a delay or, more precisely, a phase shift of the signal. A delay negatively affects the prediction and also the stability, which is not desirable. It is important to find a balance between how hard you want to filter the measurement value and how much delay you can allow [1, 61].

Filtration can be accomplished with a first-order low-pass filter. If a stronger filtering is needed than a first-order filter, two first-order filters can be connected in series to create a second-order filter.

The low-pass RC filter of the first order in Figure 4.4 section 4.1 is described by

$$T_f\, V'_{out}(t) = -V_{out}(t) + V_{in}(t)$$

If the designations are changed, the

$$T_f\, y_f'(t) = -y_f(t) + y(t)$$

Where y is the unfiltered measurement signal and $y_f$ is the filtered measurement signal. $T_f$ is the time constant of the filter.

Since the computer cannot differentiate, the differentiation must be approximated in order to be programmed. An approximation of the derivative in the equation above is obtained by using the backward difference as shown below.

$$T_f \frac{y_f(t_k) - y_f(t_{k-1})}{h} \approx -y_f(t_k) + y(t_k)$$

The above can be rewritten to the below by dividing by Acting

$$\frac{y_f(t_k) - y_f(t_{k-1})}{h} = -\frac{1}{T_f} y_f(t_k) + \frac{1}{T_f} y(t_k)$$

Multiply by h and simplify away h on the left side

$$y_f(t_k) - y_f(t_{k-1}) = -\frac{h}{T_f} y_f(t_k) + \frac{h}{T_f} y(t_k)$$

Gather $y_f(t_k)$ on the left side

$$y_f(t_k) + \frac{h}{T_f} y_f(t_k) = y_f(t_{k-1}) + \frac{h}{T_f} y(t_k)$$

Which can be rewritten as

$$\left(1 + \frac{h}{T_f}\right) y_f(t_k) = y_f(t_{k-1}) + \frac{h}{T_f} y(t_k)$$

Divide by $1 + \frac{h}{T_f}$ to get the below first-order filter equation

$$y_f(t_k) = \frac{y_f(t_{k-1})}{1 + \frac{h}{T_f}} + \frac{\frac{h}{T_f}}{1 + \frac{h}{T_f}} y(t_k)$$

To save CPU power, the equation above can be rewritten to form two parts. The parameter equation 4.8 is executed only if any variable in p1 is changed [6, p. 144].

$$p1 = \frac{1}{1 + \frac{h}{T_f}} \quad (4.8)$$

$$y_f(t_k) = p1\, y_f(t_{k-1}) + (1 - p1) y(t_k) \quad (4.9)$$

### 4.2.5 Seamless transition between automatic and manual

A controller can be in either off, automatic or manual mode. Automation means that the PID algorithm continuously calculates and updates the automatic control signal uauto. In manual mode, an operator can select the value of the manual control signal uman. When switching from automatic to manual and from manual to automatic, it is desirable that the outgoing control signal uout does not jump from one value to another.

When switching from automatic to manual or continuous in automatic mode, the manual value of the control signal $u_{is\ assigned}$ to the value of the control signal uauto just before the changeover, so that there is no change in the output control signal uout. This solution is a simplified version of what is described in [5, p. 425, F13.15].

In manual mode, the PID algorithm is executed as if the controller were running automatically, but the uauto control signal is not outsourced to the ECU. It is the value of the uman that the operator specifies that uout assumes, which in turn manipulates the ECU, i.e. $u_{out}$ is assigned to the uman. To get the PID algorithm to follow the manual value posted on uout, the anti-windup function described in section 4.2.3 can be used. By assigning uauto to uout, the unlimited control signal v will follow uout. This means that no change of the outgoing control signal is possible when switching from manual to automatic [5, p. 425, F13.15].

### 4.2.6 Jump-free parameter change

If a controller parameter is changed, e.g. the gain K, it is desirable that the outgoing control signal $u_{out}$ does not jump from one value to another. According to [5, p. 425], this can be achieved by changing the state of the I-part according to the equation below, in the event of a parameter change.

$$I_{new} = I_{old} + K_{old}(b_{old}\, y_{sp} - y) - K_{new}(b_{new}\, y_{sp} - y) \quad (4.10)$$

### 4.2.7 Policy code

By putting together the different parts of Figure 4.9, a policy code is obtained according to principles from [5, p. 428, F13.19] and [8, p. 19]. In order to be able to easily derive the equations, the equations number Ex.x and figure number Fx.x are given.

The code in Figure 4.13 is taken from [5, p. 428, F13.19]. The author of this report's contribution is to derive and step-by-step explain the background to the code in Figure 4.13.

In Figure 4.9, the filter and the D part are separated because it is desirable to be able to study/log the filtered measured value. In order for the regulator's response time to be as short as possible, it is important that the control signal is exhibited as soon as it is calculated [8, p. 16].

```
calculate parameters only when settings are changed
p11=1/(1+h/Tf)      filter constant E4.8
p13=K*h/Ti          Integral Gain E4.3
p14=h/Tt            anti-windup gain E4.4
p15=-K*Td/h         derivative gain E4.6

Bumpless parameter changes
I=I+Kold*(bold*ysp-y)-Knew*(bnew*ysp-y) E4.10

control algorithm
while(true) {
   y=getProcessVariable()     get process variable
   P=K*(b*ysp-y)              proportional party E4.1
   yf=p11*yfold+(1-p11)*y     filter process variable E4.9
   D=p15*(yf-yfold)           derivative party E4.7
   v=P+I+D                    compute nominal output F3.10
   u=saturateOutput(v, uLow, uHigh) // saturate output E4.2
   setManipulatedVariable(u) // set manipulated variable
   I=I+p13*(ysp-y)+p14*(u-v)  integral part E4.5
   yfold=yf                   update yfold
   wait(h)                    wait h seconds
}
```

**Figure 4.9** Policy code for a PID controller where the measured value is filtered in a first-order filter before the D section. Between `getProcessVariable()` and `setManipulatedVariable(u)`, 5 multiplications, 3 subtractions, and 3 additions are used.

It is possible to slightly reduce the time between `the getProcessVariable()` and `setManipulatedVariable(u)` functions by splitting the filter into two parts and moving `the p11*yfold` calculation to after `setManipulatedVariable(u).`
The filter `yf=p11*yfold+(1-p11)*y` is thus divided into two parts
`yf=Ftemp+(1-p11)*y`
`Ftemp=p11*yfold`
The parenthesis in the D-part `D=p15*(yf-yfold)` can be removed and then
`D=p15*yf-yfold*p15`
The D part can be divided into two parts
`D=p15*yf-Dtemp`
`Dtemp=yfold*p15`
The rewriting of the D-part in Figure 4.10 means that the variable `yfold` is no longer needed.

```
calculate parameters only when settings are changed
p11=1/(1+h/Tf)      filter constant E4.8
p13=K*h/Ti          Integral Gain E4.3
p14=h/Tt            anti-windup gain E4.4
p15=-K*Td/h         derivative gain E4.6

Bumpless parameter changes
I=I+Kold*(bold*ysp-y)-Knew*(bnew*ysp-y) E4.10

control algorithm
while(true) {
   y=getProcessVariable()     get process variable
   P=K*(b*ysp-y)              proportional party E4.1
   yf=Ftemp+(1-p11)*y         filter process variable
   D=p15*yf-Dtemp             derivative party
   v=P+I+D                    compute nominal output F3.10
   u=saturateOutput(v, uLow, uHigh) // saturate output E4.2
   setManipulatedVariable(u) // set manipulated variable
   I=I+p13*(ysp-y)+p14*(u-v)  integral part E4.5
   yfold=yf                   update yfold
   Ftemp=p11*yf               update part of the filter
   Dtemp=yf*p15               update part of derivative part
   wait(h)                    wait h seconds
}
```

**Figure 4.10** Policy code for a PID controller where the measured value is filtered in a first-order filter before the D section. Between `the getProcessVariable()` and `setManipulatedVariable(u)` functions, 4 multiplications, 3 subtractions, and 3 additions are used.

### 4.2.8 Code with minimal latency

If the D part and the filter are dialed the same, it is possible to minimize the time between `getProcessVariable()` and `setManipulatedVariable(u)` and thus minimize controller response time to a minimum [5, p. 428, F13.18]. A derivation of the code in [5, p. 428, F13.18] is given below.

As mentioned earlier in Section 4.2.4, the continuous derivative part of and $D(t) = -K\,T_d\,y'(t)$ the discrete-time form of the D part of

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

Section 4.2.4.1 described the equation for a digital low-pass filter.
If the designations are changed, the
$T_f\, D_f'(t) = -D_f(t) + D(t)$

Where D is the unfiltered derivative portion and $D_{f\ is}$ the filtered derivative portion. $T_f$ is the time constant of the filter. Filtering the measurement signal or the value of the D-part gives the same end result.

A discrete-time equation of the above filter equation is obtained by using backward difference

$$T_f \frac{D_f(t_k) - D_f(t_{k-1})}{h} \approx -D_f(t_k) + D(t_k)$$

Which can be rewritten by adding Df(tk) to both sides of the equal sign and simplifying

$$D_f(t_k) + T_f \frac{D_f(t_k) - D_f(t_{k-1})}{h} = D(t_k)$$

It is possible to rewrite the equations into an equation where the filter and the D-part are integrated. The D(tk) of the D-part is substituted by the filter equation.

$$D_f(t_k) + T_f \frac{D_f(t_k) - D_f(t_{k-1})}{h} = -K\ T_d \frac{y(t_k) - y(t_{k-1})}{h}$$

If the fractions are shared,

$$D_f(t_k) + T_f\left(\frac{D_f(t_k)}{h} - \frac{D_f(t_{k-1})}{h}\right) = -K\ T_d\left(\frac{y(t_k)}{h} - \frac{y(t_{k-1})}{h}\right)$$

The parentheses around the fractions are removed by multiplying Tf and –K Td by the fractions inside the parentheses.

$$D_f(t_k) + T_f \frac{D_f(t_k)}{h} - T_f \frac{D_f(t_{k-1})}{h} = -K\ T_d \frac{y(t_k)}{h} + KT_d \frac{y(t_{k-1})}{h}$$

If $T_f \frac{D_f(t_{k-1})}{h}$ added to both sides of the equal sign, the

$$D_f(t_k) + T_f \frac{D_f(t_k)}{h} = T_f \frac{D_f(t_{k-1})}{h} - K\ T_d \frac{y(t_k)}{h} + K\ T_d \frac{y(t_{k-1})}{h}$$

Because $D_f(t_k) + T_f \frac{D_f(t_k)}{h} = \left(1 + \frac{T_f}{h}\right) D_f(t_k)$ you get

$$\left(1 + \frac{T_f}{h}\right) D_f(t_k) = T_f \frac{D_f(t_{k-1})}{h} - K\ T_d \frac{y(t_k)}{h} + K\ T_d \frac{y(t_{k-1})}{h}$$

To abbreviate h in the denominators, both sides are multiplied by h.

$$h\left(1 + \frac{T_f}{h}\right) D_f(t_k) = T_f \frac{D_f(t_{k-1})}{h} h - K\ T_d \frac{y(t_k)}{h} h + K\ T_d \frac{y(t_{k-1})}{h} h$$

After all the hs that can have been abbreviated and the h is multiplied into the parenthesis of the left side, the

$$(h + T_f)D_f(t_k) = T_f\, D_f(t_{k-1}) - K\, T_d\, y(t_k) + K\, T_d\, y(t_{k-1})$$

To get Df(tk) alone on the left side, both ranks are divided by h + Tf and abbreviated

$$D_f(t_k) = \frac{T_f\, D_f(t_{k-1})}{h + T_f} - \frac{K\, T_d y(t_k)}{h + T_f} + \frac{K\, T_d\, y(t_{k-1})}{h + T_f}$$

To save CPU power, the equation above can be rewritten to form three parts. The parameter equations 4.11 and 4.12 are executed only if any variable in the equations changes.

$$p3 = \frac{T_f}{h + T_f} \quad (4.11)$$

$$p24 = \frac{K\, T_d}{h + T_f} \quad (4.12)$$

$$D_f(t_k) = p3\, D_f(t_{k-1}) - p24\, y(t_k) + p24\, y(t_{k-1}) \quad (4.13)$$

Figure 4.11 provides a first draft of the code, including the equations above.

```
calculate parameters only when settings are changed
p3=Tf/(Tf+h)         filter constant E4.11
p24=K*Td/(h+Tf)      derivative gain E4.12
p5=K*h/Ti            Integral Gain E4.3
p6=h/Tt              anti-windup gain E4.4

Bumpless parameter changes
I=I+Kold*(bold*ysp-y)-Knew*(bnew*ysp-y) E4.10

control algorithm
while(true) {
   y=getProcessVariable()     get process variable
   P=K*(b*ysp-y)              proportional party E4.1
   Df=p3*Dfold-p24*y+p24*yold D-part and filter E.13
   v=P+I+Df                   compute nominal output F3.10
   u=saturateOutput(v, uLow, uHigh) // saturate output E4.2
   setManipulatedVariable(u) // set manipulated variable
   I=I+p5*(ysp-y)+p6*(u-v)    integral part E4.5
   Dfold=Df                   update Dfold
   yold=y                     update yold
   wait(h)                    wait h seconds
}
```

**Figure 4.11** First draft of a PID controller where the D-part and a first-order filter are merged. Between `getProcessVariable()` and `setManipulatedVariable(u)`, 5 multiplications, 2 subtractions, and 3 additions are used.

The equation `Df=p3*Dfold-p24*y+p24*yold can` be divided into two parts, then you get

`Df=x-p24*y` (4.14)

`x=p3*Dfold+p24*yold` (4.15)

The rewriting of the D-part + filter in Figure 4.12 means that the variable `Dfold` and `Yold` are no longer needed.

```
calculate parameters only when settings are changed
p3=Tf/(Tf+h)         filter constant E4.11
p24=K*Td/(h+Tf)      derivative gain E4.12
p5=K*h/Ti            Integral Gain E4.3
p6=h/Tt              anti-windup gain E4.4

Bumpless parameter changes
I=I+Kold*(bold*ysp-y)-Knew*(bnew*ysp-y) E4.10

control algorithm
while(true) {
   y=getProcessVariable()     get process variable
   P=K*(b*ysp-y)              proportional party E4.1
   Df=x-p24*y                 part of D-part & filter E4.14
   v=P+I+Df                   compute nominal output F3.10
   u=saturateOutput(v, uLow, uHigh) // saturate output E4.2
   setManipulatedVariable(u) // set manipulated variable
   Dfold=Df                   update Dfold
   yold=y                     update yold
   x=p3*Df+p24*y              part of D-part & filter E4.14
   I=I+p5*(ysp-y)+p6*(u-v)    integral part E4.5
   wait(h)                    wait h seconds
}
```

**Figure 4.12** Second draft of a PID controller where the D-part and a first-order filter are merged. Between `getProcessVariable()` and `setManipulatedVariable(u)`, 3 multiplications, 2 subtractions, and 2 additions are used.

If the variable Df in the equation 4.14 `x=p3*Df+p24*y` is substituted by `x-p24*y` , we get

```
x=P3*(X-P24*Y)+P24*Y
```

The parentheses are removed by multiplying in p3

```
x=P3*X-P3*P24*Y+P24*Y
```

By breakout, the above equation can be rewritten into

```
x=p3*x+p24*(1-p3)*y
```

The parameter p24 is replaced by p4.

p4

```
x=p3*x+p24*(1-p3)*y
```

Which can be rewritten to

`x=p3*x+p4*y` (4.16)

The parameter p4 is given by the following where `p3` is substituted by `Tf/(Tf+h)`

```
p4=K*Td/(h+Tf)*(1-Tf/(Tf+h))
```

Since `1=(Tf+h)/(Tf+h)`, the equation above can be rewritten and then simplified to

```
p4=K*Td/(h+Tf)*((Tf+h)/(Tf+h)-Tf/(Tf+h))
p4=K*Td/(h+Tf)*((Tf+h-Tf)/(Tf+h))
p4=K*Td*h/(h+Tf)*(Tf+h)
```
(4.17)

The parentheses in the equation 4.1 `P=K*(b*ysp-y)` can be removed by multiplying K into the parentheses, then you get

```
P=K*b*ysp-K*y
```

The variable `p24` in the equation `Df=x-p24*y` is replaced by `K*Td/(h+Tf)`

```
Df=x-K*Td/(h+Tf)*y
```

The variables `P` and `Df` in the equation `v=P+I+Df` from Figure 4.11 can be replaced by `K*b*ysp-K*y` and `x-K*Td/(h+Tf)` respectively, giving

```
v=K*b*ysp-K*y+I+x-K*Td/(h+Tf)*y
v=K*b*ysp-(K+K*Td/(Tf+h))*y+x+I
```

To save CPU power, the equation above can be rewritten to form three parts. Parameter equations 4.18 and 4.19 are executed only if any variable in the equations is changed [5, p. 428, F13.19].

```
p1=K*b
```
(4.18)

```
p2=K+K*Td/(Tf+h)
```
(4.19)

```
v=p1*ysp-p2*y+x+I
```
(4.20)

In Figure 4.13, the above equations are used to realize a PID controller with a short response time.

```
calculate parameters only when settings are changed
p1=K*b              set-point gain E4.18
p2=K+K*Td/(Tf+h)     PD gain E4.19
p3=Tf/(Tf+h)        filter constant E4.11
p4=K*Td*h/(Tf+h)*(Tf+h) // derivative gain E4.17
p5=K*h/Ti           Integral Gain E4.3
p6=h/Tt             anti-windup gain E4.4

Bumpless parameter changes
I=I+Kold*(bold*ysp-y)-Knew*(bnew*ysp-y) E4.10

control algorithm
while(true) {
   y=getProcessVariable()     get process variable
   v=p1*ysp-p2*y+x+I          compute nominal output 4.20
   u=saturateOutput(v, uLow, uHigh) // saturate output E4.2
   setManipulatedVariable(u) // set manipulated variable
   x=p3*x+p4*y                part of D-part & filter E4.16
   I=I+p5*(ysp-y)+p6*(u-v)    integral part E4.5
   wait(h)                    wait h seconds
}
```

**Figure 4.13** A PID controller where the D-part and a first-order filter are fused. The controller has a low latency because the number of operations between `getProcessVariable()` and `setManipulatedVariable(u)` is minimal, 2 multiplications, 1 subtraction, and 2 additions [5, p. 428, F13.18].

# 5 Regulator Setting

The purpose of this chapter is to give the reader the knowledge to use the AMIGO (Approximate M-constrained integral gain optimization) method to find suitable regulator parameters. Selecting the correct regulator parameters is important in order for the controller to meet the following requirements [7]:

- Reduce load disturbances.
- Measurement noise must have little effect.
- The output signal must follow changes in the setpoint.
- The closed system must not be sensitive to variations in the characteristics of the process.

## *5.1 Controller Parameters*

Table 5.1 lists the parameters that must be specified before a PID controller can be used.

| Parameter | Description |
|---|---|
| ulow | Minimum value of the control signal u |
| uhigh | Maximum value of the control signal u |
| YSP | Setpoint, sp is an abbreviation for "set point" |
| Tt | Tracking time constant, e.g. in seconds |
| h | Sampling time in e.g. seconds |
| Tanalog f | Time constant in e.g. seconds for the analogue measured value filter |
| Acting | The time constant of the derivative filter in e.g. seconds |
| K | Reinforcement |
| Ti | Integration time in e.g. seconds |
| Td | The derivative time in e.g. seconds |
| b | Setpoint weighting factor, value 0-1 |

**Table 5.1** Regulator parameter.

**ulow, uhigh:** Indicates the minimum and maximum permissible values of the control signal.

**ysp:** The setpoint, i.e. the value that the regulator should follow.

**Tt:** Time constant in e.g. seconds that controls how quickly the integral part should be restored after a limitation of the control signal. $T_t$ is referred to as "tracking time constant" in English. The time constant Tt should be greater than Td but less than Ti. A rule of thumb is to choose Tt according to $T_t = \sqrt{T_i T_d}$ [5, p. 80] for a PID controller and for a PI controller Tt = 0.5 $T_{in}$ [7, p. 5].

**h:** The sampling time is the time in e.g. seconds between each execution of the regulator algorithm. Sampling rate = 1/h. A rule of thumb is to choose the sampling time shorter than one-fifth of the process's time constant T [1, p. 75].

**Tanalog f:** The time constant of the analogue measured value filter in e.g. seconds. To avoid the alias effect, the filter must eliminate any frequencies higher than half the sampling rate [5, p. 414]. The time constant $T_{analogue\ f}$ shall be chosen so that the frequencies above half the sampling frequency are attenuated, e.g. 16 times [5, p. 414].

**Acting:** The time constant of the derivative filter in e.g. seconds. Depending on the implementation, this filter filters the measured value before the D-part or D-part output. The filter is necessary for the prediction of the D-part to be reliable even if the measurement signal contains noise, see Figure 3.12.

According to [1, p. 58], the rule of thumb for the D filter Tf = Td / 10 is a good rule to a serial PID controller but not a parallel one. A more detailed discussion can be found in [1, p. 59] in the following:
A.J. Isaksson and S.F. Graebe, Derivative filter is an integral part of PID design, IEE Proceedings - Control Theory and Applications, January 2002, Volume 149, Issue 1, p. 41-45 and B. Lennartsson, Reglertekniks grunder, 4th edition, Studentlitteratur 2002.

According to [5, p. 73], Tf can be chosen according to Tf = Td/N, where typical values of N are between 2 and 20.

The disadvantage of filtering according to [1, p. 60] is that the filter causes a delay or, more precisely, a phase shift of the signal. A delay negatively affects the D-part prediction function, which is not desirable. It is important to find a balance between how hard you want to filter and how much delay you can allow [1, 60].

The derivative filter reduces the robustness of a PID controller, but it is possible to compensate for the dynamics of the filter, see section 5.3.2.

**K:** The gain K is set to a positive value if the measured value goes in the same direction as the control signal, otherwise negative. For example, the value of K can be obtained by using the AMIGO method.

**Ti:** The integration time in e.g. seconds. For example, the value of $T_i$ can be obtained by using the AMIGO method.

**Td:** The derivative time in e.g. seconds. For example, the value of $T_d$ can be obtained by using the AMIGO method.

**b:** Setpoint weighting factor, value 0-1. For example, the value of b can be obtained by using the AMIGO method.

## *5.2 Process Analytics*

In order to be able to select the controller parameters K, Ti, Td and b correctly, information is needed about the dynamic properties of the process, i.e. how the measurement signal of the process reacts when the control signal to the process varies. A mathematical process model can be obtained by setting up a differential equation based on physical laws. This differential equation is then transformed into a transfer function using Laplace transform. If the transfer function of the process is known, the parameters of a PID controller can be calculated by using several different methods, such as pole placement. This approach requires university-level knowledge of physics and mathematics. An alternative is to perform experiments on a process to obtain process data and to apply e.g. the AMIGO method to obtain the controller parameters K, Ti, Td and b. The disadvantage of only conducting experiments is that the process must be available and that it is not possible to simulate the control before the process is built. Not all processes are suitable for experimentation because they are safety-critical, such as the process of flying an aircraft on autopilot. In the engineering and process industries, it is often possible to conduct experiments on the process. The following sections describe several different types of experiments.

### 5.2.1 Step-response analysis on stable processes

By manually inserting the controller and making a step change to the control signal, the figures below can be obtained 5.1 and 5.2.

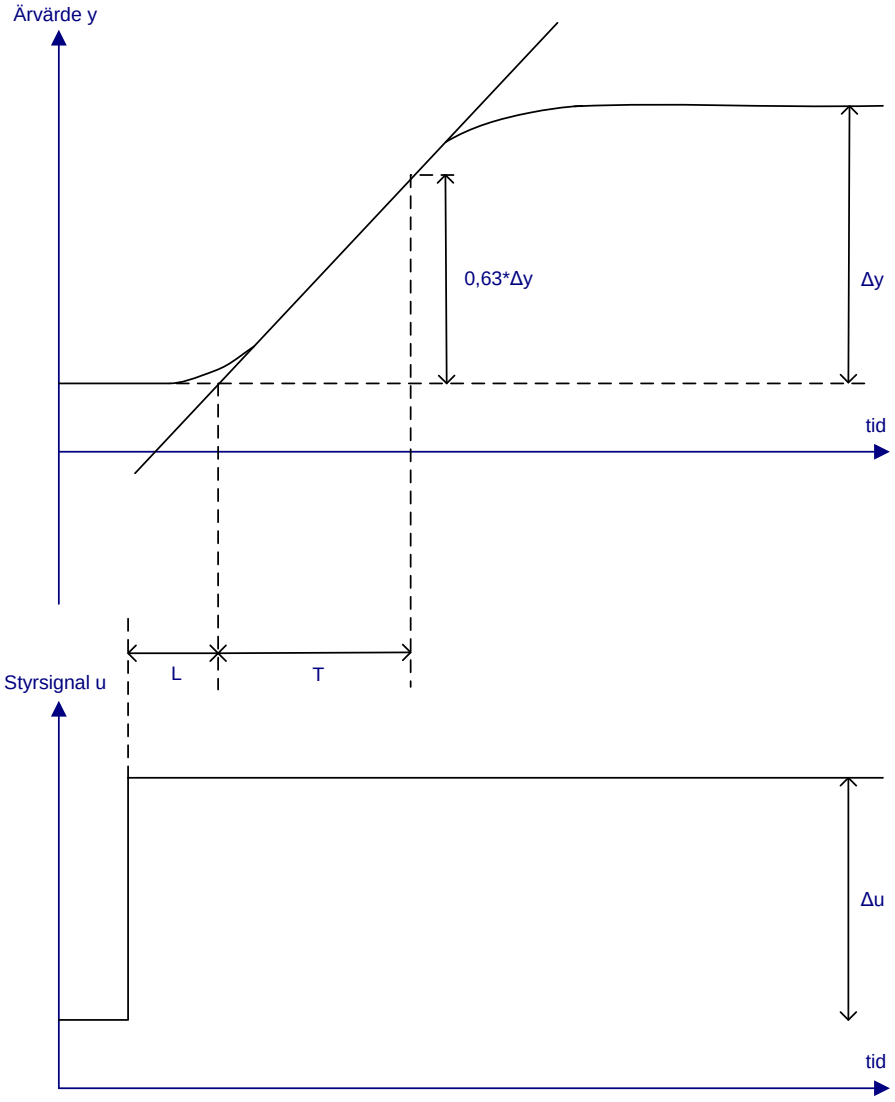


**Figure 5.1** Step responses from a non-integrative process[5, p. 49].

The static gain Kp is given by $K_p = \frac{\Delta y}{\Delta u}$. A tangent following the steepest slope of the step response is drawn in the figure, see Figure 5.1. The dead time L is given at the point where the tangent intersects the stable measured value before the control signal step was performed. The time constant T is the time it takes for the measurement signal to reach 63% of its delta value. However, the dead time is not included in this time, see Figure 5.1 [5, p. 48]. The time $T_{63}$ is defined as $T_{63} = T + L$.

### 5.2.2 Step Response Analysis on Integrating Processes

If a step response is performed on an integrative process, the measured value is not stabilized, see Figure 5.2.

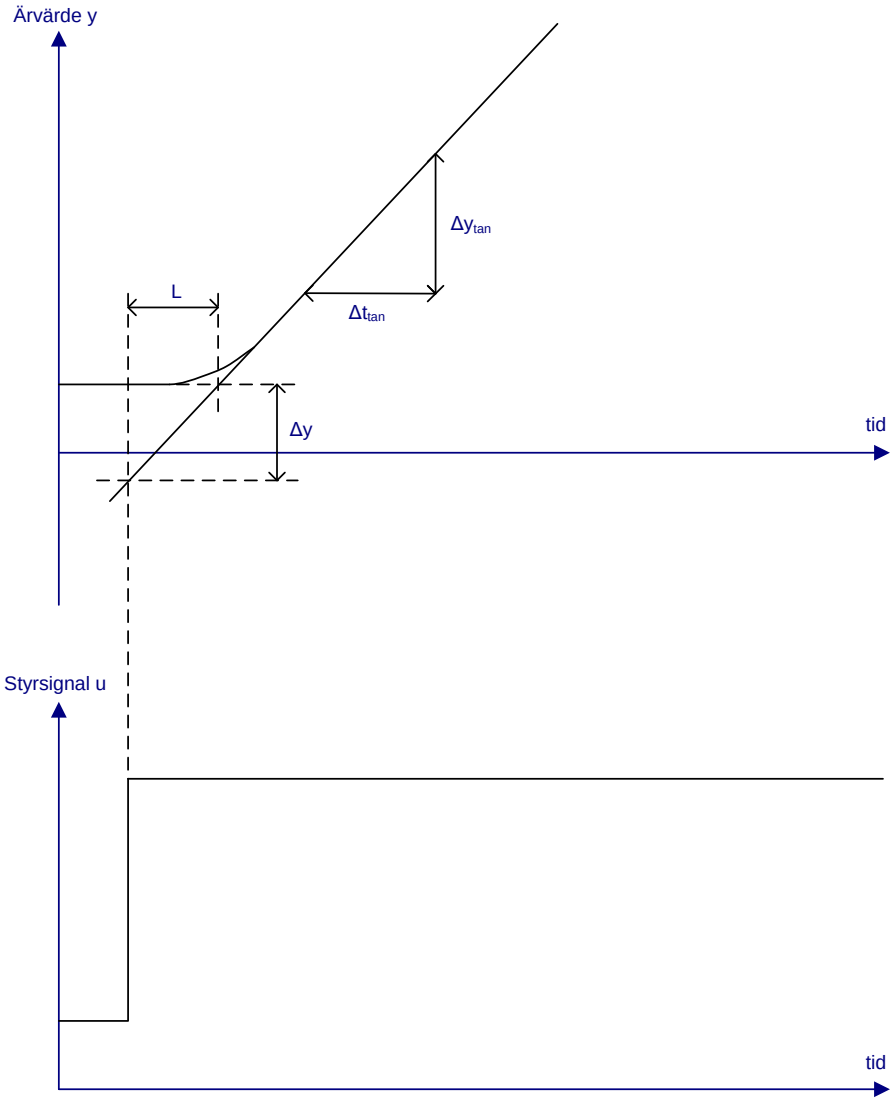


**Figure 5.2** Step-by-step responses from an integrative process[5, p. 49][2, p. 62].

A tangent following the steepest slope of the step response is drawn in the figure, see Figure 5.2. The velocity gain Kv is given by the coefficient of direction on the tangent, i.e.

$K_v = \frac{\Delta y}{L}$ or $K_v = \frac{\Delta y_{\tan}}{\Delta t_{\tan}}$. The dead time L is given at the point where the tangent intersects the stable measured value before the control signal step was performed [5, p. 49]. Note that the static gain Kp cannot be measured using a step response from an integrative process.

An example of an integrating process is a level control in a tank. If the tank is filled with as much as it is drained, the level will remain constant. If more is filled in the tank than is drained, the level will rise. If less is filled into the tank than is drained, the level will drop.

### 5.2.3 Double pulse analysis

A variant of the step response analysis is to subject the process with a double pulse, as illustrated in Figure 5.3.

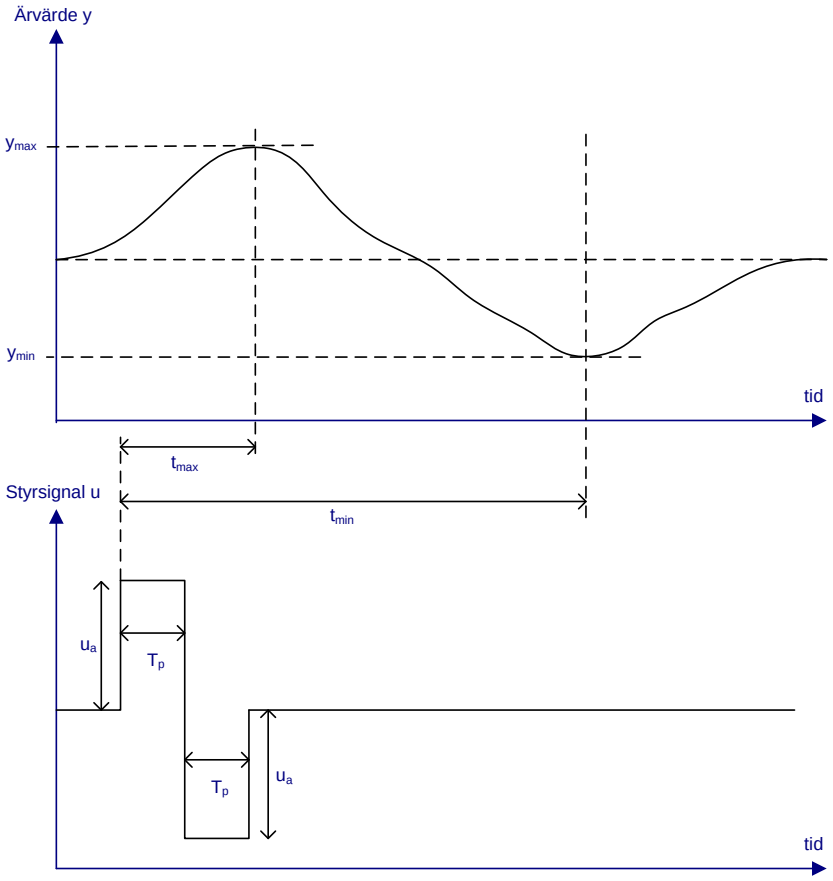


**Figure 5.3** Double pulse analysis [5, p. 50].

The amplitude of the pulse ua is chosen so that it is significantly greater than the noise. The pulse width Tp is selected a little longer than the time delay of the process. The measured values $y_{max}$, ymin, and the times tmax and tmin are read, see Figure 5.3. One advantage of the method is that the measurement value returns to its original value, another that the time required is shorter than the step response analysis because you do not have to wait for the measurement value to stabilize. The disadvantages of the method are that it is difficult to determine the min and max accurately and that the estimation of the static gain Kp is poor. The time $T_{63}$ is defined as $T_{63} = T + L$.

Using the equations below, the static gain Kp, the time constant T and the dead time L can be calculated. The dead time L can be calculated in two ways as shown below. This can be used to determine whether the double pulse analysis is applicable to the process being experimented with [5, p. 50]

$$K_p = -\frac{y^2_{\max}}{u_a y_{\min}}$$

$$T = -\frac{T_p}{\log(1 + y_{\max}/y_{\min})}$$

$$L = t_{\max} - T_p$$

$$L = t_{\min} - 2T_p$$

### 5.2.4 Frequency Response Analysis by Self-oscillation

In frequency response analysis by self-oscillation, a P-controller is allowed to control the process. The gain K of this regulator is increased until the control circuit begins to oscillate at the ultimate frequency, see Figure 5.4.

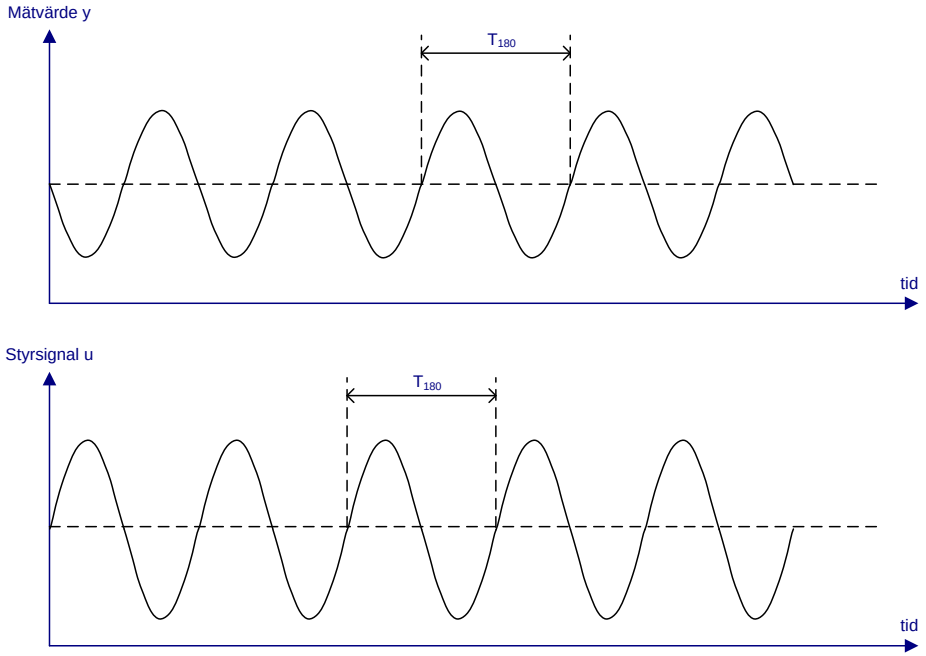


**Figure 5.4** Frequency response analysis by self-oscillation.

At the ultimate frequency where the control signal and the measurement signal are phased by 180°, the period time T180 is read and the regulator's gain is read K. The regulator gain at the ultimate frequency is called the ultimate gain and is denoted Ku. The process gain K180 at the ultimate frequency can be calculated according to

$$K_{180} = \frac{1}{K_u}$$

The ultimate frequency when the phase shift is 180° is denoted ω180 (Greek letter omega), measured in radians per second and given by

$$\omega_{180} = \frac{2\pi}{T_{180}}$$

A disadvantage of the method is that it balances on the stability limit. Another major disadvantage is that the process is greatly affected by the experiment [5, p. 26][5, p. 52][5, p. 161][2, p. 66].

### 5.2.5 Frequency Response Analysis with Sine Signal

In frequency response analysis with a sine signal, a sine wave signal is used to control the control signal u, see Figure 5.5.

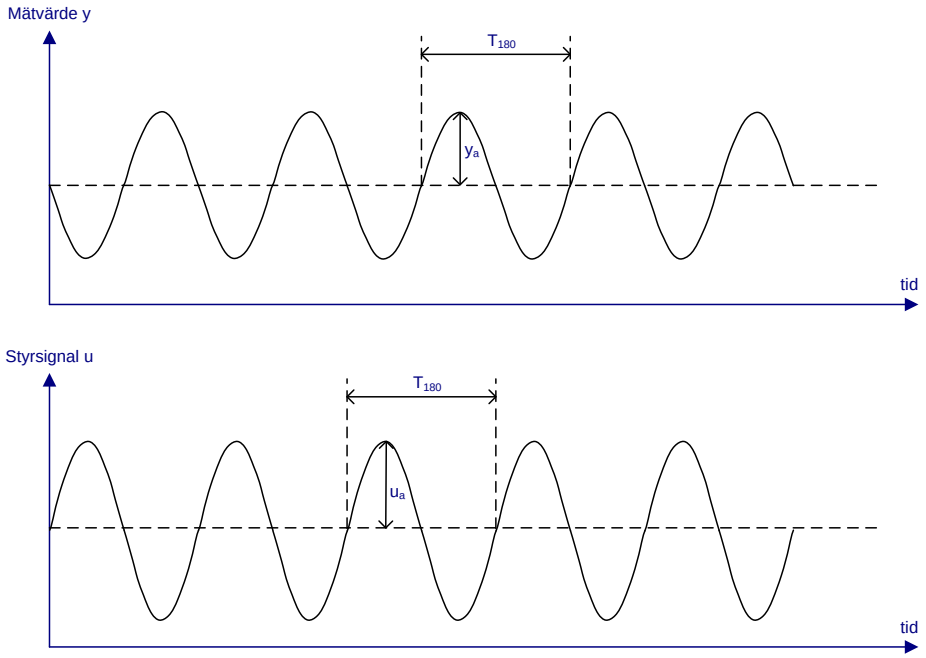


**Figure 5.5** Frequency response analysis with sine signal.

At the ultimate frequency where the control signal and the test signal are phased by 180°, the period time T180, the amplitude of the measurement signal $y_a$ and the amplitude of the control signal ua, see Figure 5.5, are read. The process gain K180 at the ultimate frequency can be calculated according to

$$K_{180} = \frac{y_a}{u_a}$$

The ultimate frequency when the phase shift is 180° is denoted ω180 (Greek letter omega), measured in radians and given by

$$\omega_{180} = \frac{2\pi}{T_{180}}$$

The regulator gain at the ultimate frequency is called the ultimate gain and is denoted Ku. The ultimate gain Ku is calculated according to

$$K_u = \frac{1}{K_{180}}$$

A disadvantage of the method is that you have to try several different frequencies to find the ultimate frequency [5, p. 21][5, p. 52][5, p. 161][2, p. 17].

### 5.2.6 Frequency response analysis using the relay method

The relay method means that the process is controlled with an on-off regulator with hysteresis, see Figure 5.6 [2, p. 124].

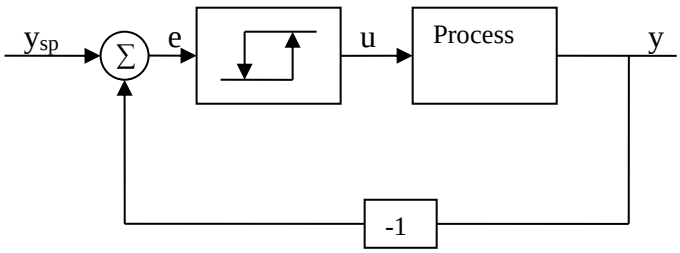


**Figure 5.6** The principle of the relay method [5, p. 53].

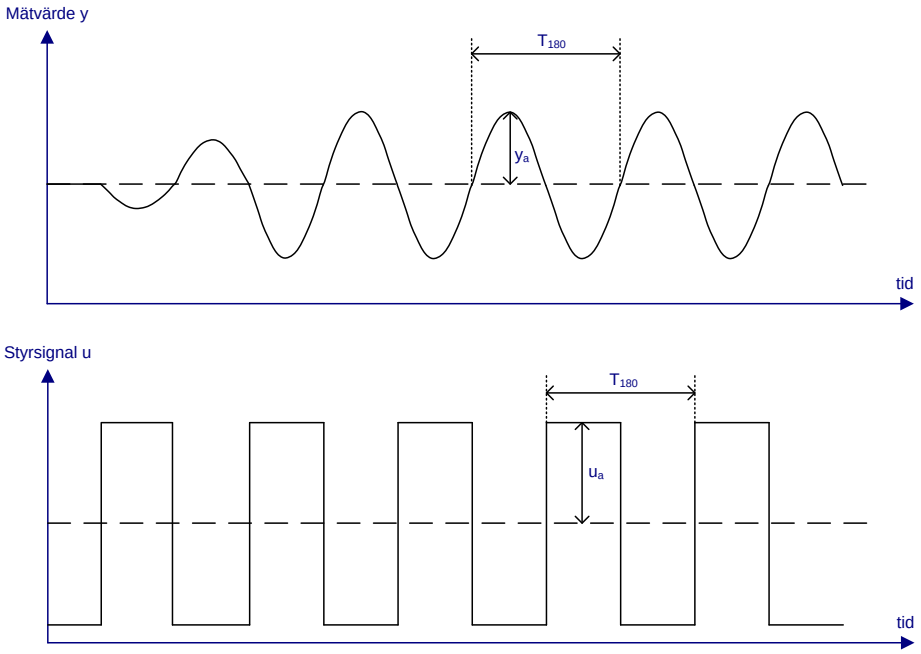


**Figure 5.7** Process analysis using the relay method [5, p. 53].

The on-off control generates a control signal u that will shift between two levels and the measured value y will oscillate around the setpoint, see figure 5.7. The oscillation has approximately the same frequency as in the frequency response analysis by self-oscillation, i.e. the ultimate frequency where the control signal and the measurement signal are phase-turned 180°. The period time T180, the amplitude of the measuring signal $y_a$ and the amplitude of the control signal ua at the ultimate frequency can be read as shown in Figure 5.7. The process gain K180 at the ultimate frequency can be calculated according to

$$K_{180} = \frac{\pi y_a}{4u_a}$$

The ultimate frequency when the phase shift is 180° is denoted ω180 (Greek letter omega), measured in radians and given by

$$\omega_{180} = \frac{2\pi}{T_{180}}$$

The regulator gain at the ultimate frequency is called the ultimate gain and is denoted Ku. The ultimate gain Ku is calculated according to

$$K_u = \frac{1}{K_{180}}$$

The relay method has two major advantages over frequency response analysis by self-oscillation. Firstly, you don't balance on the stability limit because the on-off regulator regulates the process. Secondly, the amplitude of the control signal can be selected so that the process is not disturbed so much. This is a great advantage because the control deviation from the desired setpoint can be reduced. Other advantages include ease of automation and short time [2, p. 124][5, p. 53][5, p. 161].

### 5.2.7 Combined Step Response Analysis and Frequency Response Analysis

The results of a step response analysis and a frequency analysis can be combined. The input data for the calculation from a step response analysis is the static gain Kp, the time constant T and the dead time L. Data from a frequency analysis is the period time T180 and the process gain K180. Using the equations below, T1, T2 and L1 are calculated, which are then used in the AMIGO method to find suitable regulator parameters [5, p. 54]. See also MATLAB code to solve these equations in Figure 5.12. Since the value of the static gain Kp is required, the controller parameters for processes that are of the integrative type cannot be calculated using the equations below.

First, the time T63 is calculated, the ultimate frequency ω180 in radians and the ultimate gain Ku.

$$T_{63} = T + L$$

$$\omega_{180} = \frac{2\pi}{T_{180}}$$

$$K_u = \frac{1}{K_{180}}$$

According to [5, p. 51], it can be assumed that T2 ≤ T1. If $T_2 = T1$, α = 1 because α = $T_2$ / T1. If both T2 and T1 are positive numbers and $T_2 < T_1$ are α < 1, but at the same time it is important to α ≥ 0. Thus, α can vary between 0 and 1.

The variables e1, e2, and e3 in the equations below are equal to zero when a perfect value of α is found. To add e1, e2, and e3 into a value, the equation can be used. To solve the equations below, a number of values of α between 0 and 1 can be tested. When e is at its smallest, an appropriate value of α has been found, and thus also T1, T2 and L1.

$$e = \sqrt{e_1^2 + e_2^2 + e_3^2}$$

$$T_1 = \frac{1}{\left(\alpha\, \omega_{180}\sqrt{2}\right)} * \sqrt{\left(\sqrt{4\alpha^2\, K_p^2\, K_u^2 + \left(1-\alpha^2\right)^2}\right) - 1 - \alpha^2}$$

$$T_2 = \alpha\, T_1$$

$$L_1 = \frac{\left(\pi - \arctan\left(\omega_{180} T_1\right) - \arctan\left(\omega_{180} T_2\right)\right)}{\omega_{180}}$$

$$e_1 = 0{,}37 - \frac{T_1}{\left(T_1 - T_2\right)} e^{-\left(T_{63} - L_1\right)/T_1} - \frac{T_2}{\left(T_2 - T_1\right)} e^{-\left(T_{63} - L_1\right)/T_2} \quad \text{To be used if } \alpha \neq 1$$

$$e_1 = 1 - e^{-\left(T_{63} - L_1\right)/T_1} - \frac{T_{63}}{T_1} e^{-\left(T_{63} - L_1\right)/T_1} - 0{,}63 \quad \text{Used if } \alpha = 1$$

$$e_2 = \left(1 + \omega_{180}^2\ T_1^2\right)\left(1 + \omega_{180}^2\ T_2^2\right) - K_p^2\ K_u^2$$

$$e_3 = \arctan\left(\omega_{180} T_1\right) + \arctan\left(\omega_{180} T_2\right) + \omega_{180}\, L_1 - \pi$$

## *5.3 The AMIGO method*

The AMIGO (Approximate M-constrained integral gain optimization) method was developed as a replacement for the Ziegler-Nichols methods from the 1940s. According to [5, p. 225], Ziegler-Nichols' methods are seriously flawed. With the help of the AMIGO method, it is possible to find suitable controller parameters with step response analysis and/or frequency analysis as input. The goal of the development of AMIGO has been to find a method that can be used for both manual and automatic setting of the controller parameters K, Ti, Td and b. The method has been developed by first applying the technique of "Robust loop shaping" and MIGO (M-constrained integral gain optimization) method to 134 test processes that are representative of the process industry. The optimal control parameters have been determined using the MIGO method. The relationship between the optimal control parameters and data from step response analyses and frequency analyses has been analyzed and the result was the AMIGO method.

AMIGO maximizes integral gain ki = K / Ti while selecting the control parameters so that the control is robust, i.e. insensitive to process variations. Setpoint tracking is handled by using setpoint weighting. In order for the control to be robust, AMIGO also provides rules for how the controller parameters should be modified if the measurement signal contains noise that cannot be filtered out, for more information on this see the section "7.9 Detuning" in [5] on page 253.

To roughly characterize the dynamics of processes, the measure relative time delay $\tau$ = L / (L+T) (Greek letter tau) is used, which assumes values between 0 and 1. Processes with a low value of are $\tau$ called lag-dominated, processes with $\tau$ close to 1 are called delay-dominated, and processes with $\tau$ around 0.5 are called balanced[5, p. 225][5, p. 262].

To sum up, T > L for processes that are lag-dominant, L > T for dead-time dominant, T = L for balanced. For a description of the dead time L and the time constant T, see Section 5.2.1.

### 5.3.1 PI or PID controller?

Analyses of the regulator parameters of the 134 processes show that the derivative part only gives a slight improvement if the process is of the balanced or downtime-dominant type. On the other hand, the derivatives section provides major improvements for lag-dominant processes [5, p. 226]. The reason why the derivative

part gives better regulator performance is that the gain k and the integral gain ki can be increased [2, p. 35]. By comparing the integral gain given by $k_i = K/Ti$ between a PI and a PID regulator, the utility of the derivative part can be assessed [5, p. 264].

In the first instance, a PI controller should be chosen because a PID controller is sensitive to measured noise [1, p. 63].

### 5.3.2 Adjustment for the dynamics of the derivative filter

The derivative filter reduces the robustness of a PID controller, but it is possible to compensate for the filter's dynamics. The AMIGO method provides the following rules on how data from the process analyses should be modified to compensate for the filter.

The step-response analysis data shall be adjusted according to

$Kp_{\text{with filter}} = Kp_{\text{without filter}}$ (no change)

Tmed filter = Tutan filter + Tf / 2

Lmed filter = Lutan filter + Tf/2

Data from a combined step-response analysis and frequency analysis shall be adjusted

$T1_{\text{with filter}} = T1_{\text{without filter}}$ (no change)

T2 with filter = T2 without filter + Tf / 2

$L1_{\text{with filter}} = L1_{\text{without filter}}$ + Tf / 2

The filter suppresses measured noise at the expense of poorer attenuation of reading disturbances. The reason for this is that the control parameters need to be adjusted so that the controller becomes slower in order to remain robust even with a filter [5, p. 251][5, p. 265].

### 5.3.3 PI parameters, step response analysis

The AMIGO equations below give the parameters K and Ti that correspond very well for all test processes with the values given by the MIGO method. The input data to the equations is data from a step-response analysis. Parameters for non-integrative processes are given by [5, p. 228]

$$K = \frac{0{,}15}{K_p} + \left(0{,}35 - \frac{L\,T}{(L+T)^2}\right)\frac{T}{(K_p L)}$$

$$T_i = 0{,}35\,L + \frac{13\,L\,T^2}{\left(T^2 + 12\,L\,T + 7\,L^2\right)}$$

Parameters for processes that are integrative are given by [5, p. 229]

$$K = \frac{0{,}35}{K_v L}$$

$$T_i = 13{,}4\,L$$

The setpoint weighting parameter b depends on the relative time lag of the process $\tau$ as [5, p. 230].

b = 0 for $\tau \leq 0{,}3$

b = 1 for $\tau > 0{,}3$

### 5.3.4 PID parameters, step response analysis

The input data to the equations is data from a step-response analysis. Parameters for non-integrative processes are given by [5, p. 233]

$$K = \frac{1}{K_p}\left(0{,}2 + 0{,}45\frac{T}{L}\right)$$

$$T_i = \frac{\left(0{,}4\ L + 0{,}8\ T\right)}{\left(L + 0{,}1\ T\right)}L$$

$$T_d = \frac{0{,}5\ L\ T}{0{,}3\ L + T}$$

For processes where $\tau > 0.3$, the AMIGO equation above gives a K that follows the K that MIGO gives well. For smaller values, the $\tau$ AMIGO K underestimates for almost all test processes.

Integration time is well described for processes where $\tau > 0.2$. For smaller values $\tau$, the AMIGO overestimates Ti for almost all processes compared to MIGO.

The derivative time Td is well described for processes where $\tau > 0.5$. In the interval $0.3 < \tau < 0.5$, the AMIGO derivative time can be half the value from MIGO. If derivative time values are used within the range $0.3 < \tau < 0.5$, robustness is impaired. For values $\tau < 0.3$, AMIGO gives a derivative time that is sometimes less than and sometimes greater than what MIGO gives [5, p. 233].

The equations give conservative parameters of $\tau < 0.5$, interpretation of [5, p. 231]. The reason for the conservative parameters is above all reduced gain K and increased integration time Ti [5, 233]. The equations can also be used for lag-dominant processes provided that conservative parameters can be accepted [5, 264].

An observation made by the author of this report is that the equations give good results for processes of the type balanced or dead-time-dominant, which do not benefit much from the derivative part, and less good values for lag-dominant processes that benefit greatly from the derivative part.

Parameters for processes that are integrative are given by

$K = 0{,}45/K_v$

$T_i = 8\ L$

$T_d = 0{,}5\ L$

The relationship between b and $\tau$ for the different processes is not very good, but a conservative rule is to choose the setpoint weighting parameter b according to [5, p. 235]

b = 0 for $\tau \leq 0{,}5$

b = 1 for $\tau > 0{,}5$

### 5.3.5 PI parameters, frequency analysis

The equations below give parameters that are reasonably close to the optimal MIGO parameters, for processes where the amplification ratio κ > 0.2 (Greek letter kappa). The amplification ratio is given by $\kappa = K_{180} / Kp$. The equations can be applied to processes that are of the balanced or deadtime-dominant type, but are unsuitable for processes that are of the lag-dominated type. The input data to the equations is data from a frequency response analysis for measured values on the process amplification K180 and the period time T180 [5, p. 239].

$$K = 0{,}16 / K_{180}$$

$$T_i = \frac{1}{(1 + 4{,}5\,\kappa)} T_{180}$$

Since data on the static gain Kp is required, a step-response analysis must also be performed to obtain a measurement value of Kp [9]. Since the value of the static gain Kp is required, controller parameters for processes that are of the integrative type cannot be calculated using the above equations.

### 5.3.6 PID Parameters, Frequency Analysis

The equations below give parameters that are reasonably close to the optimal MIGO parameters, for processes where the amplification ratio κ > 0.2 (Greek letter kappa). The amplification ratio is given by $\kappa = K_{180} / Kp$. The equations can be applied to processes that are of the balanced or deadtime-dominant type, but are unsuitable for processes that are of the lag-dominated type. The input data to the equations is data from a frequency response analysis for measured values on the process amplification K180 and the period time T180 [5, p. 241].

$$K = \frac{(0{,}3 - 0{,}1\,\kappa^4)}{K_{180}}$$

$$T_i = \frac{0{,}6}{(1 + 2\,\kappa)} T_{180}$$

$$T_d = \frac{0{,}15(1 - \kappa)}{(1 - 0{,}95\,\kappa)} T_{180}$$

Since data on the static gain Kp is required, a step-response analysis must also be performed to obtain a measurement value of Kp [9]. Since the value of the static gain Kp is required, controller parameters for processes that are of the integrative type cannot be calculated using the above equations.

An observation made by the author of this report is that the equations are useful for the balanced or dead-time-dominant process, which is not very useful for the derivatives part, and not useful for lag-dominant processes that are very useful for the derivative part.

### 5.3.7 PID parameters, step response analysis and frequency analysis

#### 5.3.7.1 Processes that are stable

The equations below give similar values for balanced or deadtime-dominant processes as in the equations from sections 5.3.4 and 5.3.6, but for lag-dominant, significantly better parameters are obtained. The parameters Kp, T1, T2 and L1 are inputs to the equations, these parameters are the result of a combined step response analysis and frequency response analysis, see Section 5.2.7.

Since the value of the static gain Kp is required in Section 5.2.7, regulator parameters for processes that are of the integrative type cannot be calculated using the equations below.

The integral gain is given by ki = K/Ti and the derivative amplification is given by kd = K Td. Controller parameters for non-integrative processes are given by [5, p. 245]

$$K = \frac{\left(\alpha_1 + \alpha_2 \frac{T_1}{L_1} + \alpha_3 \frac{T_2}{L_1} + \alpha_4 \frac{T_1\, T_2}{{L_1}^2}\right)}{K_p}$$

$$k_i = \frac{\left(\beta_1 \frac{1}{L_1} + \beta_2 \frac{T_1}{{L_1}^2} + \beta_3 \frac{T_2}{{L_1}^2} + \beta_4 \frac{T_1\, T_2}{{L_1}^3}\right)}{K_p}$$

$$k_d = \frac{\left(\left(\gamma_1 L_1 + \gamma_2 T_1 + \gamma_3 T_2 + \gamma_4 \frac{T_1\, T_2}{L_1}\right)\frac{(T_1 + T_2)}{(T_1 + T_2 + L_1)}\right)}{K_p}$$

Parameters of the equations above

| | | | |
|---|---|---|---|
| α1 = 0,19 | α2 = 0,37 | α3 = 0,18 | α4 = 0,02 |
| β1 = 0,48 | β2 = 0,03 | β3 = -0,0007 | β4 = 0,0012 |
| γ1 = 0,29 | γ2 = 0,16 | γ3 = 0,20 | γ4 = 0,28 |

The Greek letters are α = alpha, β = beta, and γ = gamma.

The integration time is given by

$$T_i = \frac{K}{k_i}$$

The derivative time is given by

$$T_d = \frac{k_d}{K}$$

In the examples [5, pp. 247 – 250], a value of b is given for the PID AMIGO step+frequency, but it is not clear when b should be 0 and 1 respectively. According to [9], the same rules as for the PID AMIGO step shall be used as set out below.

b = 0 for $\tau \leq 0{,}5$

b = 1 for $\tau > 0{,}5$

### 5.3.7.2 Processes that are of the integrative type

According to [9], the authors of [5] have not given much thought to how to experimentally produce the parameters Kv, T2 and L1, which are input data to the equations below.

One way according to [9] is to generate a pulse response, i.e. to submit a short pulse with a duration of t and an altitude of 1/t, following the same principles as in section 5.2.1. The shorter the t, the better. In order for the pulse to be considered a pulse, it must be significantly shorter, it is L and T that are estimated in the model. There's no sharp limit, but if t is less than one-tenth of L and T, it should work fine. This means that the response looks like a step response from a stable process. From this, the three parameters can be determined according to Kv=Kp, T2=T, L1=L.

If Kv and L are estimated with a step response analysis according to section 5.2.2, a model is obtained where L summarizes both the true dead time and the time constant, in a step response analysis according to section 5.2.1. The relation is L(step response) = L(pulse response) + T(pulse response) [9].

According to [9], the double pulse analysis from section 5.2.3 may be possible, but it needs to be further investigated.

The integral gain is given by ki = K/Ti and the derivative amplification is given by kd = K Td. Controller parameters for processes that are integrative are given by [5, p. 246]

$$K = \frac{\left(\alpha_2 \frac{1}{L_1} + \alpha_4 \frac{T_2}{L_1^2}\right)}{K_v}$$

$$k_i = \frac{\left(\beta_2 \frac{1}{L_1^2} + \beta_4 \frac{T_2}{L_1^3}\right)}{K_v}$$

$$k_d = \frac{\left(\gamma_2 + \gamma_4 \frac{T_2}{L_1}\right)}{K_v}$$

The integration time is given by

$$T_i = \frac{K}{k_i}$$

The derivative time is given by

$$T_d = \frac{k_d}{K}$$

Parameters of the equations above

| | | | |
|---|---|---|---|
| α1 = 0,19 | α2 = 0,37 | α3 = 0,18 | α4 = 0,02 |
| β1 = 0,48 | β2 = 0,03 | β3 = -0,0007 | β4 = 0,0012 |
| γ1 = 0,29 | γ2 = 0,16 | γ3 = 0,20 | γ4 = 0,28 |

The Greek letters are α = alpha, β = beta, and γ = gamma.

In the examples [5, pp. 247 – 250], a value of b is given for the PID AMIGO step+frequency, but it is not clear when b should be 0 and 1 respectively. According to [9], the same rules as for the PID AMIGO step shall be used as set out below.

b = 0 for $\tau \leq 0{,}5$

b = 1 for $\tau > 0{,}5$

## *5.4 AMIGO example*

This section provides an example of how controller parameters can be selected using the AMIGO method. The example is taken from [5, p. 247]. The MATLAB codes are written by the author of this report.

### 5.4.1 Example of a lag-dominant process

Measurement data from a step-response analysis on a lag-dominant process that is not integrative shows that the dead time L = 0.075, the time constant T = 1.04 and the static gain Kp = 1. A frequency analysis shows that the process gain K180 = 0.0091 and the period time T180 = 0.199.

#### 5.4.1.1 PI - Step Response Analysis

MATLAB code that calculates PI parameters with input from a step response analysis is given in Figure 5.8.

```
format short;
L=0.075;
T=1.04;
Kp=1;
K=0.15/Kp+(0.35-L*T/(L+T)^2)*T/(Kp*L)
Ti=0.35*L+13*L*T^2/(T^2+12*L*T+7*L^2)
tau=L/(L+T);
if tau > 0.3
  b=1
else
  b=0
End
ki=K/Ti

K = 4.1333
Ti = 0.5389
b = 0
ki = 7.6696
```

**Figure 5.8** MATLAB code based on equations from Section 5.3.3.

### 5.4.1.2 PID - Step Response Analysis

MATLAB code that calculates PID parameters with input from a step-response analysis is given in Figure 5.9.

```
format short;
L=0.075;
T=1.04;
Kp=1;
K=1/Kp*(0.2+0.45*T/L)
Ti=(0.4*L+0.8*T)/(L+0.1*T)*L
Td=0.5*L*T/(0.3*L+T)
tau=L/(L+T);
IF TAU > 0.5
  b=1
else
  b=0
End
ki=K/Ti
IF TAU < 0.5
  info = 'Warning conservative parameters'
End

K = 6.4400
Ti = 0.3612
Td = 0.0367
b = 0
ki = 17.8308
info = Warning conservative parameters
```

**Figure 5.9** MATLAB code based on equations from Section 5.3.4.

### 5.4.1.3 PI - Frequency Analysis

The MATLAB code that calculates PID parameters with input data from a frequency analysis is given in Figure 5.10.

```
format short;
K180=0.0091;
T180=0.199;
Kp=1;
kappa=K180/Kp
K=0.16/K180
Ti=1/(1+4.5*coat)*T180
b='?'
ki=K/Ti
if kappa <= 0.2
  info = 'Warning parameters are not appropriate'
End

cloak = 0.0091
K = 17.5824
Tue = 0.1912
b = ?
ki = 91.9719
info = Warning parameters are not appropriate
```

**Figure 5.10** MATLAB code based on equations from Section 5.3.5.

### 5.4.1.4 PID - Frequency Analysis

The MATLAB code that calculates PID parameters with input from a frequency analysis is given in Figure 5.11.

```
format short;
K180=0.0091;
T180=0.199;
Kp=1;
kappa=K180/Kp;
K=(0.3-0.1*kappa^4)/K180
Ti=0.6/(1+2*kappa)*T180
Td=0.15*(1-kappa)/(1-0.95*kappa)*T180
b='?'
ki=K/Ti
if kappa <= 0.2
  info = 'Warning parameters are not appropriate'
End

K = 32.9670
Ti = 0.1173
Td = 0.0298
b = ?
ki = 281.1309
info = Warning parameters are not appropriate
```

**Figure 5.11** MATLAB code based on equations from Section 5.3.6

### 5.4.1.5 PID – Combined Step Response Analysis and Frequency Analysis

Parameters from a combined step response analysis and frequency analysis are calculated in two steps.

#### *5.4.1.5.1 $T_1$, $T_2$ and L1*

The MATLAB code in Figure 5.12 calculates T1, T2 and L1, which are then used in the AMIGO method to calculate the appropriate regulator parameters. The input data for the calculation is the static gain Kp, the time constant T, the dead time L, the period time T180 and the process gain K180.

```
format short;
Kp=1;
T=1.04;
L=0.075;
T63=T+L;
T180=0.199;
K180=0.0091;
omega180=2*pi/T180;
Ku=1/K180;
step=0.00001;
startAlpha=0;
endAlpha=1;
alpha=startAlpha+step;
bestE=100000000;
bestT1=0;
bestT2=0;
bestL1=0;

while alpha <= endAlpha
  T1=1/(alpha*omega180*sqrt(2))*sqrt((sqrt(4*alpha^2*Kp^2*Ku^2+(1-
alpha^2)^2))-1-alpha^2);
  T2=alpha*T1;
  L1=(pi-atan(omega180*T1)-atan(omega180*T2))/omega180;
  if alpha~=1
    e1=0.37-T1/(T1-T2)*exp(-(T63-L1)/T1)-T2/(T2-T1)*exp(-(T63-
L1)/T2);
  else
    e1=1-exp(-(T63-L1)/T1)-T63/T1*exp(-(T63-L1)/T1)-0.63;
  End
  e2=(1+omega180^2*T1^2)*(1+omega180^2*T2^2)-Kp^2*Ku^2;
  e3=atan(omega180*T1)+atan(omega180*T2)+omega180*L1-pi;
  e=sqrt(e1^2+e2^2+e3^2);
  if e < bestE
    bestE=e;
    bestT1=T1;
    bestT2=T2;
    bestL1=L1;
  End
  alpha=alpha+step;
End

bestE, bestT1, bestT2, bestL1
bestE = 1.5161e-007
bestT1 = 0.9988
bestT2 = 0.1057
bestL1 = 0.0102
```

**Figure 5.12** MATLAB code based on equations from Section 5.2.7.

#### *5.4.1.5.2 Controller parameters*

The MATLAB code that calculates PID parameters with input from a combined step response analysis and frequency analysis is given in Figure 5.13.

```
format short;
T1=0.9988;
T2=0.1057;
L1=0.0102;
Kp=1;
L=0.075;
T=1.04;
alpha1=0.19;
alpha2=0.37;
alpha3=0.18;
alpha4=0.02;
beta1=0.48;
beta2=0.03;
beta3=-0.0007;
beta4=0.0012;
gamma1=0.29;
gamma2=0.16;
gamma3=0.20;
gamma4=0.28;
K=(alpha1+alpha2*T1/L1+alpha3*T2/L1+alpha4*T1*T2/L1^2)/Kp
ki=(beta1*1/L1+beta2*T1/L1^2+beta3*T2/L1^2+beta4*T1*T2/L1^3)/K
p;
Ti=K/ki
kd=((gamma1*L1+gamma2*T1+gamma3*T2+gamma4*T1*T2/L1)*(T1+T2)/(T
1+T2+L1))/Kp;
Td=kd/K
tau=L/(L+T);
IF TAU > 0.5
  b=1
else
  b=0
End
Ki

K = 58.5810
Ti = 0.1291
Td = 0.0521
b = 0
ki = 453.7330
```

**Figure 5.13** MATLAB code based on equations from Section 5.3.7.

### 5.4.1.6 Summary of the example

Table 5.2 summarises the parameters from the calculations performed with the MATLAB code presented in Section 5.4.1. Calculations from sections 5.4.1.3 and 5.4.1.4 are not included in Table 5.2 because $\kappa \leq 0{,}2$ which means that the values are not usable.

| Method | K | Ti | Td | b | $k_i$ |
|---|---|---|---|---|---|
| PI MIGO | 3,56 | 0,660 | | 0 | 5,39 |
| PI AMIGO step response analysis | 4,13 | 0,539 | | 0 | 7,67 |
| PID MIGO | 56,9 | 0,115 | 0,0605 | 0 | 495 |
| PID AMIGO step response analysis | 6,44 | 0,361 | 0,0367 | 0 | 17,8 |
| PID AMIGO Step Response Analysis + Frequency Response Analysis | 58.6 | 0,129 | 0,0521 | 0 | 454 |

**Table 5.2** Controller parameters from the example.

Values from the PI AMIGO step response analysis are similar to the values from the PI MIGO. Since the process is lag-dominant, conservative parameters for PID AMIGO step response analysis are obtained that are far from the optimal PID MIGO values. The values from PID AMIGO IVR + Frequency Response Analysis are consistent with PID MIGO.
Since the process in the example from section 5.4.1 is lag-dominant, the benefit of the derivative part is great because the ki value can be maximized.

# 6 Conclusions

The aim of this report was to answer the following questions:

- How does a PID controller work?
- How can a PID controller be implemented?
- How is a PID controller trimmed using the AMIGO method?

The author of this report believes that the answers to these questions have been answered in this report. Furthermore, it should be possible for e.g. a Software Engineering student to understand, design, implement and set up a PID controller with the help of this report.

Since PID controllers are very common, it is important that there are reliable methods for selecting optimal controller parameters based on simple experiments on the process. The researchers Åström, Hägglund and their students' work with the AMIGO method is a major advance, since Ziegler-Nichols' methods have major shortcomings according to [5, p. 225]. With the help of the AMIGO method, it is possible to find suitable values for the controller parameters K, Ti, Td, and b manually or automatically.

# 7 References

[1] Krister Forsman, Automatic Control for the Process Industry, Studentlitteratur, 2005

[2] Tore Hägglund, Praktisk processregler, Studentlitteratur, 1997

[3] Bertil Thomas, Modern Automatic Control, Liber, 1992

[4] Holmström and Smedhamre, Komvux matematik etapp III, Almqvist & Wiksell Förlag, 1992

[5] Karl J Åström and Tore Hägglund, Advanced PID control, ISA, 2006

[6] Gustaf Olsson and Gianguido Piani, Computer systems for automation and control, Prentice Hall, 1992

[7] K. J. Åström, Lecture 9 - PID Control, October 2002

[8] Ola Dahl, Lecture Notes Lecture 9 RT7010 – Automatic Control, Malmö University Technology and Society

[9] Tore Hägglund, E-mail correspondence between Tore Hägglund and Jonas Wahlfrid, January – February 2007