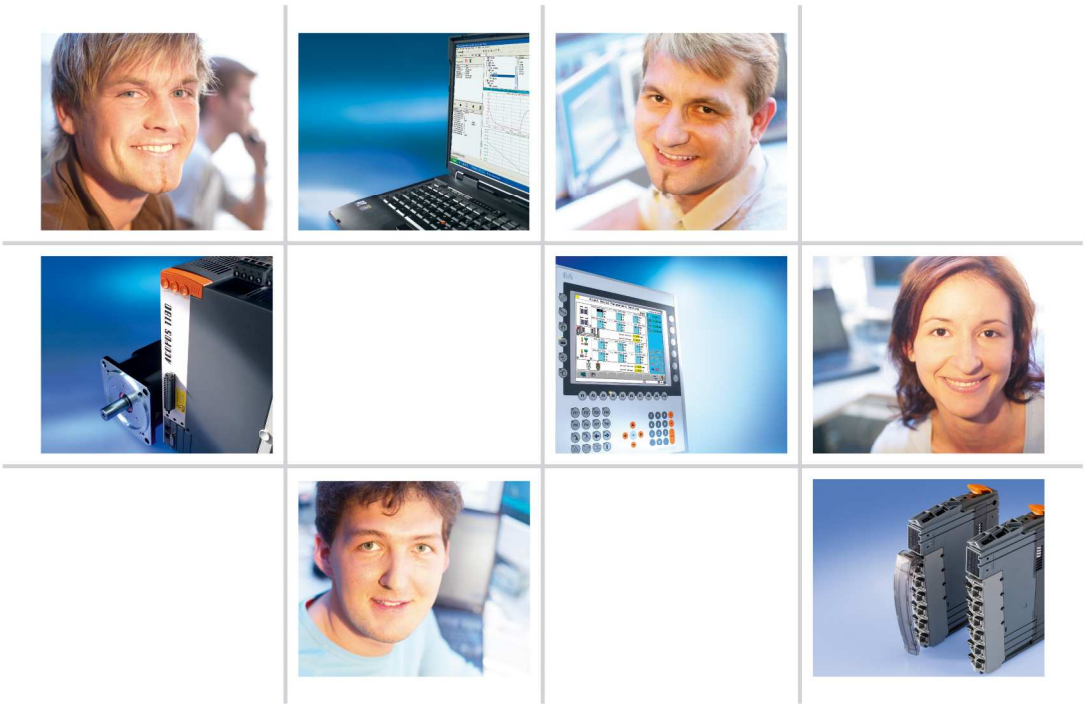
Closed Loop Control with LOOPCONR

TM261



Perfection in Automation

www.br-automation.com

Requirements

Training modules:	TM260
Software:	Automation Runtime 2.85
Hardware:	Optional B&R simulation model 4SIM.00-01

Table of contents

1. INTRODUCTION

Closed loop control is an important part of industrial technology and is usually a basic requirement for productive machines and systems as well as for high-quality products.

Closed loop control has a reputation as a "sophisticated area of expertise" because knowledge of complex mathematics is required to understand the fundamental methods.

This training module will follow a two-part approach to closed loop control based on software controllers in order to meet the wide range of demands of practitioners in the industrial field who must produce highly satisfactory results in a short amount of time, and to correspond with system analyzers who operate in a more theoretical manner:

Part I is a **practical approach**, which will accompany your training at B&R and your autodidactic practical experience. This part focuses on the quick implementation of controllers based on the Automation Studio library LOOPCONR.

With the help of different function blocks from this library, a highly effective control loop can be created and adapted to a variety of applications in a flexible manner.

Particular attention will be given to the use of integrated procedures for automatically setting controllers (autotuning).

Part II offers a compressed – more theory-oriented and still easy-to-understand – approach to the topic. The basic methods and terminology of closed loop control will be handled here. This part should not be viewed as a substitute to educational books. This is simply not possible for the reason of limited topical breadth. **Part II** can also be used as a **compendium and reference work**, which will hopefully provide explanations and ideas when a controller is not working as desired.

Throughout the entire training module, closed loop control will generally be explained using temperature controllers, which exhibit relatively simple behavior often used in the field.

1.1 Training guide objectives

Participants will be familiar with the simple basic concepts of closed loop control and will be able to manually configure a PID controller.

Participants will know how to use integrated autotuning procedures.

Participants will be able to configure the function block for pulse width modulation and know how to implement a closed loop control with opposing manipulated variables.

Participants will understand the B&R simulation model and know how to implement a closed loop control for the integrated temperature controlled system.

Participants will gain an overview of the most important function blocks in the LoopConR library.

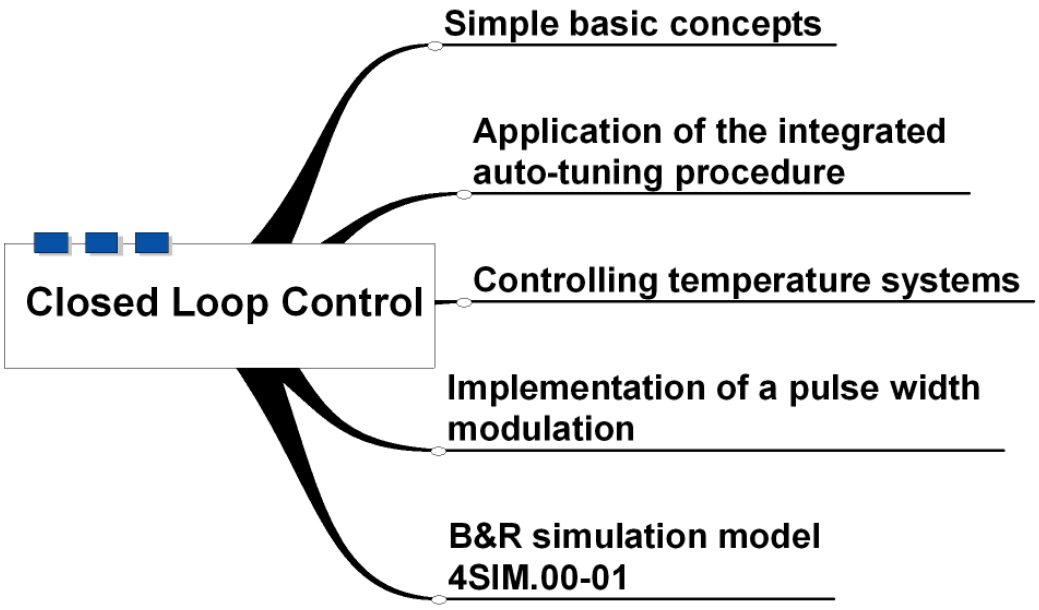


Fig. 1 Training guide overview

1.2 Compendium objectives

Participants will understand the fundamental concepts such as dynamic systems, the establishment of models and the identification of controlled systems, block diagrams, Bode diagrams and autotuning procedures.

Participants will understand the influence of dead times, measurement errors, signal sampling and modulated actuator signals.

Participants will understand advanced control structures such as pre-filter and mixed control loop as well as examples for practical application.

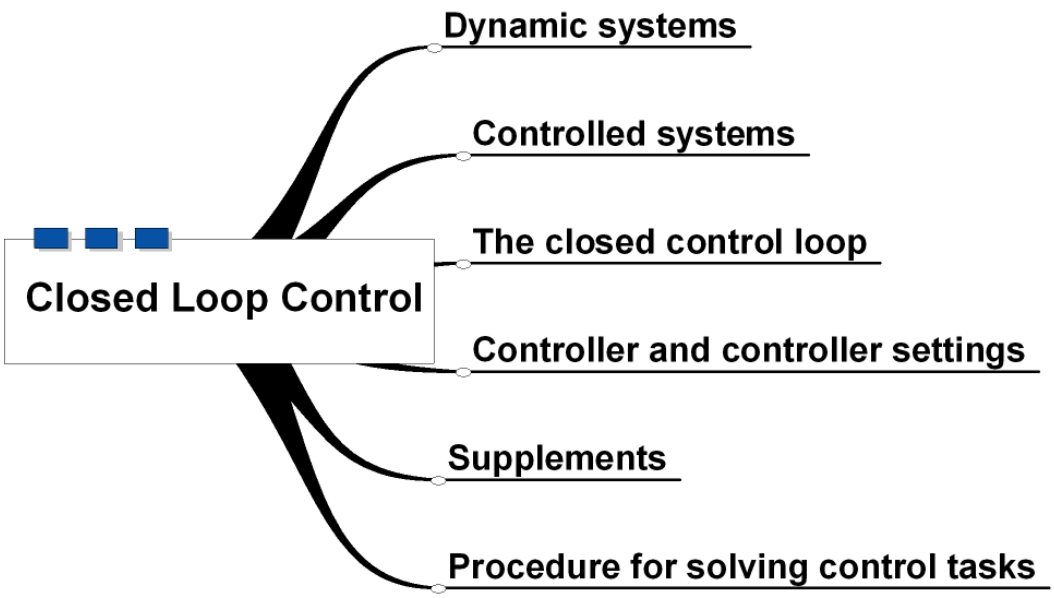


Fig. 2 Compendium overview

Notes

PART I

Training guide

(A practical approach)

2. FUNCTIONALITY OF THE LOOPCONR LIBRARY

The function blocks in the **LoopConR** library provide the following functions:

- Controller function block
- Autotuning procedure
- Modulation procedure
- Signal processing
- Simulation of thermal controlled systems
- Control of a continuous servo drive without position feedback.
- Controller function blocks and tuning process especially for temperature systems

This library can be used to cover most standard tasks in the area of closed loop control and signal processing.
Unlike the **LoopCont** library, all calculations in this library are made using only floating point arithmetic (REAL).

System requirements:

The function blocks in the LoopConR library use floating point arithmetic for calculations and can be used optimally on SG4 controllers with regard to computing time.

Since SG3 controllers do not use floating point arithmetic for calculations, cycle time violations may occur due to the floating point emulation used for the function blocks in the LoopConR library.

In this case, either use function blocks in the LoopCont library that primarily rely on fix point arithmetic or increase the cycle time of the task.

3. SIMPLE BASIC CONCEPTS

Closed loop control theory explores how to influence systems in such a way that a specific variable can posses a specified value at any time.

Room temperature provides a basic example: heating is regulated via a thermostat in such a way that the value specified on the thermostat is maintained.

Let's get started with an exercise getting to know the SlimPID() function block in order to ease your introduction into the theory of closed loop control. Solutions to the exercises can be found in the appendix of this training module.

In practice, PID controllers are very frequently used for temperature controllers. For this reason, this training module will also be demonstrating how to use these function blocks in tasks related governing temperatures.

We will establish a simple control loop in the following example. We will be operating the SlimPID() function block as true P-controller and will examine the effects of different manually defined control parameters. The LCRSimModExt() function block can be used to implement a simulation model of an extruder. Use the parameters specified in the example in the online for this.

Task: SlimPID() P-controller

Use the function blocks LCRSlimPID() and LCRSimModExt() to construct the following control loop:

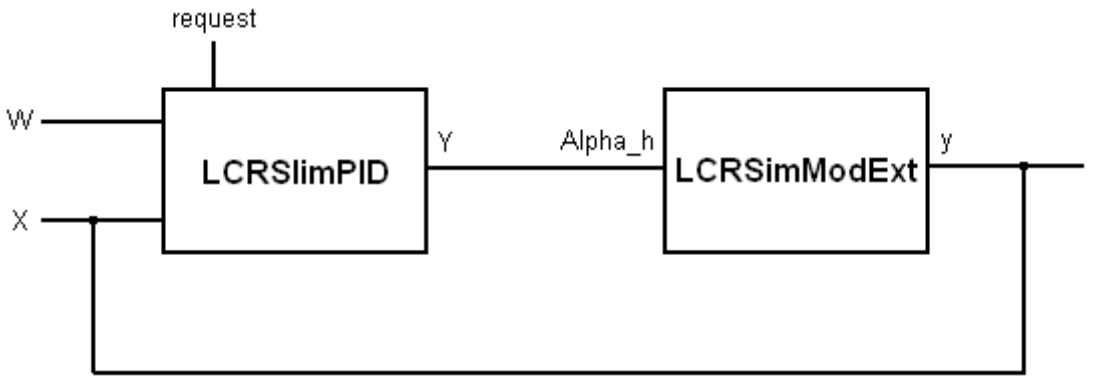


Fig. 3: Block diagram – control loop

Using the following gains:

- kp= 0.5
- kp= 3
- kp= 8

Execute set value jumps and record the set and actual temperatures and the gain using Trace.

Examine the remaining controller deviation *e* (difference between the set value and the actual value).

Examine the stability of the control loop. Oscillations occurring during compensation must fade as quickly as possible.

Which gain is best suited with regard to remaining controller deviation and a fast reduction in oscillations?

Solution approach:

In the Ladder Diagram, create the control loop described above. Set up the function block LCRSimModExt() using the parameters listed in the example in the online help.

The output value *Y* from the function block LCRSlimPID() is the manipulated variable that is fed to the *Alpha_h* input of the function block LCRSimModExt() as a heating control action. The resulting controlled variable *y* is fed back to the LCRSlimPID() function block at input *X* as the actual value.

The trace should take at least 10 minutes.

Enter the gain values for the variable *Kp*, located in the structure attached to *<LCRSlimPID-instance name>.pPar*. Set the *request* input of the LCRSlimPID() function block to *LCRSLIMPID_REQU_READ_PARAS (3)* and back to *LCRSLIMPID_REQU_OFF (0)* so that the function block applies the value for Kp (edge-controlled). Additional information about operating the LCRSlimPID() function block can be found in the online help.

3.1 What does control mean?

A closed loop control has the task of getting the output variable of a controlled system, the controlled variable *X*, to a predefined value, the reference variable *W*, and to maintain this value despite influence from disturbance variables *Z*. In a closed loop control, the actual value of the controlled variable *X* is continuously determined and is compared with the set value specified by the reference variable *W*. Unlike open loop controlling, this is a closed loop, which means that the variables (manipulated variables) that influence the process are independently established with suitable control mechanisms (actuators) from measured process variables instead of being specified externally.

The controller deviation *e* determined by comparing *W* and *X* is processed to the manipulated variable *Y* with a specific control algorithm and fed to the final controlling device.

The next figure shows the block diagram for a standard control loop with the following elements:

- Plant: the system to be controlled (process or system).
- Controlled variable (actual value): the variable to be intentionally influenced by the controller (output variable of the controlled system or actual value).
- Reference variable (set value): set value of the controlled variable (e.g. specified by operator).
- Measuring element (sensor, measuring device): provides the controller with a measurement value of the controlled variable (typically via an input module).
- Control deviation: the difference between the reference and controlled variable.
- Controller: uses the control deviation to generate a corresponding signal in order to affect the controlled system (typically via an output module).
- Actuator: the connecting element between the controller, which generally only provides weak signals, and the system to be controlled, which usually requires strong signals to have an effective influence. The output variable of the actuator is the manipulated variable.
- Disturbance variable: describes the influence of non-measurable variables that affect the control loop.

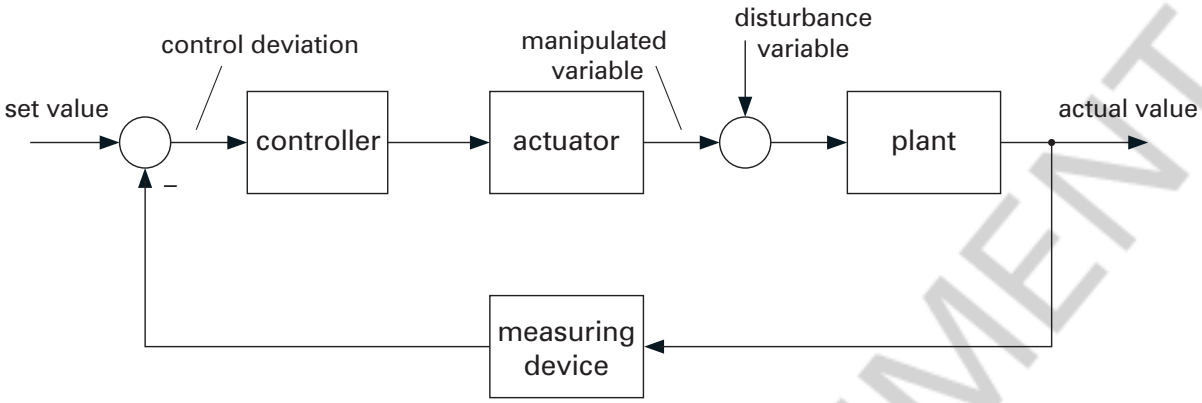


Fig. 4: Standard control loop

	Description
X	Controlled variable (actual value)
W	Reference variable (set value)
e	Control deviation e = W – X
Y	Manipulated variable
Z	Disturbance variable
R(s)	Controller transfer function
G(s)	Transfer function for the system to be controlled

Regardless of their implementation, controllers are differentiated according to their typical step responses. Different types of control behavior include P, I, PI, PD and PID.

A controller's step response is its typical reaction on the output (manipulated variable Y) to a signal jump on the input when the control loop is interrupted.

3.2 P controller behavior

With the P controller, the output variable *Y* is proportional to the controller deviation *e*. The factor kp is a proportional coefficient. The proportional coefficient kp specifies by which amount the manipulated variable *Y* will change when the controller deviation *e* is changed by a specific amount.

$$Y_P(t) = k_P \cdot e(t)$$

Thus, the controller always requires a controller deviation to adjust the actuator. A disturbance variable or reference variable, which causes a controller deviation in a control loop, can never be completely cleared with the P controller as seen in the previous exercise. This remaining control deviation is a disadvantage of the P controller. Although it is small when the kp proportional coefficients are large, kp cannot be increased infinitely. This would cause instable controller operation.

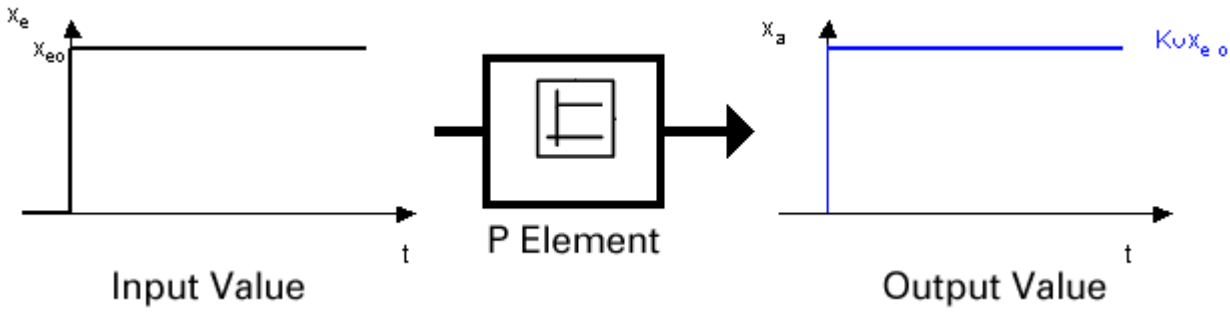


Fig. 5: Reaction of a P controller

Behavior:

- creates one of the manipulated variables proportional to the control deviation
- quick reaction to control deviations, quick rise
- never fully compensates (because a manipulated variable is not output when control deviation is missing), thus resulting in a remaining control deviation
- very simple and inexpensive (often only mechanical)

3.3 I control behavior

With the I controller, the manipulated variable *Y* is proportional to the time integral of the controller deviation.

$$Y_I(t) = \frac{k_P}{T_N} \int e(t) \cdot dt$$

The integral action time T_n is the time span, which a constant control error must meet for the I-element to generate the same manipulated variable as would be generated immediately by the P-element.

Although an I-controller reacts slowly to a change in the controller difference, the advantage is that it completely compensates for the controller difference that's always present for a P-controller.

However, an I-element lowers the stability of a control loop and causes overshoot. The smaller the integral action time, the stronger the effect of the I-element.

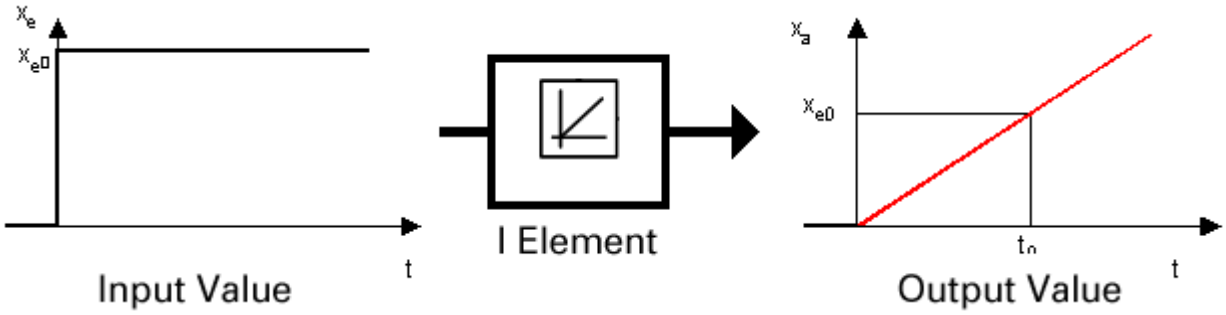


Fig. 6: Reaction of an I controller

Behavior:

- the manipulated variable changes with constant gradient at a constant control deviation
- delayed reaction to control deviations
- fully compensates for control deviations (because the manipulated variable continues to change until the control deviation is eliminated)
- tends to overshoot and lowers the stability of the control loop

3.4 PI controller

With the PI controller, the manipulated variable *Y* is equal to an addition of the output variables from a P and an I-element. The manipulated variable is first changed, just as with the P controller. A change to the manipulated variable then occurs again, which also like the I controller, is equal to the time integral of the controller deviation. Therefore, the PI controller combines the advantages of both controllers. It reacts quickly to controller deviations (P-element) and compensates them entirely (I-element).

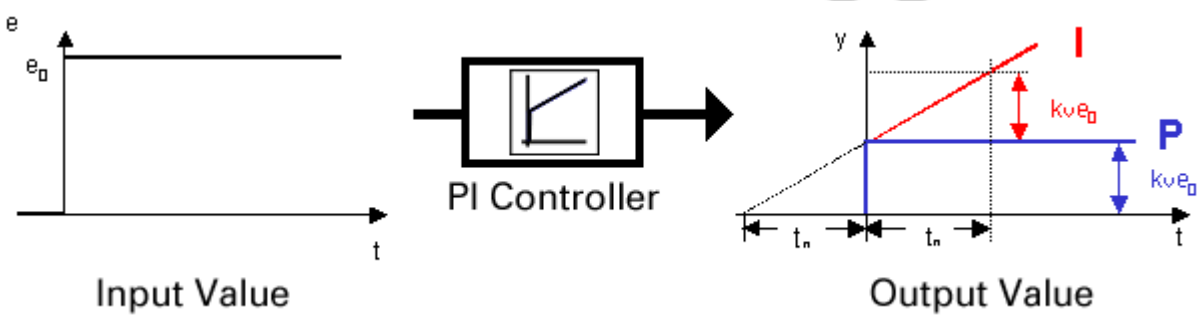


Fig. 7: Reaction of a PI controller

3.5 D controller behavior

The D-element creates a manipulated variable, which is proportional to the temporal derivative of the control deviation.

$$Y_D(t) = k_P \cdot T_V \cdot \dot{e}(t)$$

The derivative action time T_v indicates the time span, which an increasing control deviation of 0 with a constant gradient must meet for the P-element to generate the same manipulated variable as the D-element.

A D-element increases the speed and improves the stability of a control loop. A larger derivative action time increases the effect of the D-element. However, a D-element does not compensate by itself. This is why it can only be used together with another controller.

3.6 Ziegler/Nichols controller settings

If the controller isn't very well known, it is usually very difficult and time-consuming to determine suitable parameters for a PI or PID controller without sufficient experience.

Configuring the controller according to the Ziegler/Nichols method is an easy way to determine suitable controller parameters without having to know the controlled system exactly. This procedure was developed in 1942 and is based on experience gained empirically.

This procedure is done as follows:

- The controller will first be operated as a true P controller.
- The controller gain kp will be increased up to the value k_{crit}, at which point the control loop reaches its stability limits and causes continuous oscillations with constant amplitude and period.
- The period duration T_{crit} of the continuous oscillation is measured.

Controller parameters can be calculated using the following table:

	Control parameters		
Controller type	**kp**	**Tn**	**Tv**
P	0.5 k_{crit}		
PI	0.45 k_{crit}	0.85 T_{crit}	
PID	0.6 k_{crit}	0.5 T_{crit}	0.12 T_{crit}

In our next task, we will determine the parameters for a PI or PID controller using the Ziegler/Nichols method.

Task: SlimPID() controller settings according to Ziegler/Nichols

Increase the gain kp in steps starting from 0 until the control loop reaches the stability limit.
First start with a small interval (approx. 0.1 to 0.5). If the effects are minimal at first, the interval can be increased.

After each change in gain, also change the set value in the range between 150 °C and 200 °C in jumps.

The goal is to find the critical gain kp = k_{crit} that brings the control loop to the stability limit. The critical gain k_{crit} is the least amount of gain needed to keep the control value oscillation at a constant amplitude and period after a set value jump.

The critical gain k_{crit} and the period of the oscillation T_{crit} are measured and used to calculate the control parameters in the table.

Calculate the Kp and Tn parameters for a PI controller.

Calculate the Kp, Tn, and Tv parameters for a PID controller.

Solution approach:

Use the same project you used for the earlier task.

You can also begin with a gain kp = 3. You already know from the previous task that the control loop is still stable at this gain.

You have now successfully configured a controller for the controlled system, and you have probably noticed that determining the control parameters empirically requires a certain amount of patience and experience.

This procedure doesn't always have to be carried out manually, however. The LCRSlimPID() function block gives us the option of using auto-tuning. Auto-tuning determines all parameters automatically by executing a number of oscillation or a step response. You will find more detailed information on how to use it in the next training example.

In addition to the Ziegler/Nichols controller configuration that you now already know, Part II of this training module – Compendium and Reference Work – will also handle the Chien, Hrones, and Reswick methods.

4. APPLICATION OF THE INTEGRATED AUTO-TUNING PROCEDURE

An auto-tuning procedure is a sequence of intercoordinated identification and controller setting procedures that run automatically and are controlled by algorithms. It is the most convenient method of setting a controller for the user.

A stimulating input signal is first actuated on the system and the system's response is recorded. The system's transfer function is approximately determined from the comparison of these input and output signals. A P/PI/PID controller is then configured for this system in such a way so that the closed control loop exhibits the desired behavior. After setting the parameters once, these procedures will run fully automatically without intervention from the user and can be repeated at any time.

The SlimPID() function block, which was discussed earlier, provides two different autotuning procedures:

- Auto-tuning with oscillation attempt
- Auto-tuning with step response

Furthermore, the function block allows you to adjust the method for determining parameters to meet your demands. A table with the different tuning options that can be specified on the function block's *request* input can be found in the online help under *Data types and constants: Tuning options*.

4.1 Oscillation attempt with the SlimPID() function block

During the oscillation attempt, a periodic square-wave signal is used as system excitation in the closed control loop. This square-wave signal is generated by a 2-position controller (comparator), which is used in place of the closed-loop controller. This procedure can be used for setting P/PI/PID controllers and features the easiest method for setting parameters. Parameters are set using the *request* input. For example, if *request* = LCRPID_TUNE_REQU_OSCILLATE (1) is selected, then the default settings for the different oscillation attempt options are set automatically.

Default options for the oscillation attempt:

Selection options	Selection	Number
Type of tuning ➔	Oscillation attempt	1
Effective direction ➔	Positive	10
Controller settings ➔	PID	100
Controller setting procedure ➔	Ziegler / Nichols	0000
Oscillation attempts ➔	2	20000
Periods per oscillation attempt ➔	4	400000

The request is a product of the sum of the numbers.

Detailed information about the available tuning options can be found in the online help under *Data types and Constants: Tuning options*

Note:

If, for example, you would like to use tuning with 3 oscillations over 5 periods with a negative control action (increasing the manipulated variable reduces the actual value), you must set "request" to 530121.

Task: SlimPID() auto-tuning with oscillation attempt

Use your existing project and carry out auto-tuning using oscillation.

Set the request input to LCRPID_TUNE_REQU_OSCILLATE (1) to execute an oscillation attempt tuning procedure.

Solution approach:
First, set the set temperature to the operating point you will be using in the future (e.g. 150°C).

Then set the *request* input of the function block LCRSlimPID() to LCRPID_TUNE_REQU_OSCILLATE (1).

Record the set temperature, the actual temperature and the manipulated variable in Trace.

Now compare the control parameters that you defined previously with the ones determined by auto-tuning. Set the *request* input of the LCRSlimPID() function block to *LCRSLIMPID_REQU_WRITE_PARAS (4)* and back to *LCRSLIMPID_REQU_OFF (0)* so that the function block copies the control parameters to the structure connected to *pPar* (edge-controlled).

4.2 Step response with the SlimPID() function block

The step response uses a manipulated variable jump as system excitation in the open control loop and offers a multitude of possibilities for setting the controller. You can choose between the controller types P/PI/PID or design processes, disturbance rejection or set-point tracking design.

The step response is configured using the *request* input. If *request* = LCRPID_TUNE_REQU_STEPRESPONSE (2), then a step response is executed using the default settings.

Default options for the step response:

Selection options	Selection	Number
Type of tuning ➔	Step response	2
Effective direction ➔	Positive	10
Controller settings ➔	PID	100
Controller setting procedure ➔	Chien / Hrones / Reswick disturbance variable design, non-periodic	1000

The request is a product of the sum of the numbers.

Detailed information about the available tuning options can be found in the online help under *Data types and Constants: Tuning options*.

A manipulated variable jump must be specified for the step response. That means that the manipulated variable (*Y0*) at which the system is close to the operating point must be approximately known. Starting from this manipulated variable *Y0*, a step ($\Delta Y = Y1 - Y0$) to a new manipulated variable *Y1* is then executed. Suitable controller parameters can then be calculated based on the reaction of the system. It is important to ensure that ΔY is large enough to cause a significant change to the controlled variable. Otherwise it will not be possible to find any suitable control parameters.

Variables and function blocks that are used internally must be used to set *Y0 and Y1* with LCRSlimPID(). The procedure for writing internal variables can be found in the online help in the section *Function blocks and functions: LCRSlimPID()* under the heading *Access to internal structures and variables*. *Y0 and Y1* can also be set to the corresponding values in the watch for test purposes.

The slope of the controlled variable must be determined for the step response. Good filtering of the signal is essential because the controlled variable almost always has overlying noise in actual practice. In the internal substructure *<LCRSlimPID instance name>.PIDTune_inst.pOptions_step*, the filter is configured using the variable *evalNFilter*.

Step response procedure:

- *Y0* is output on the controller output until the transient effect of the system is finished and the controlled variable is close to the set value.
- *Y1* is then output on the controller output until the necessary P/PI/PID parameters have been found. This can take a different amount time depending on the design process.
  - Disturbance variable design: Tuning is complete as soon as the maximum slope of the controlled variable has been detected.
  - Reference variable design: Tuning is complete as soon as the transient effect in the new operating point is finished.

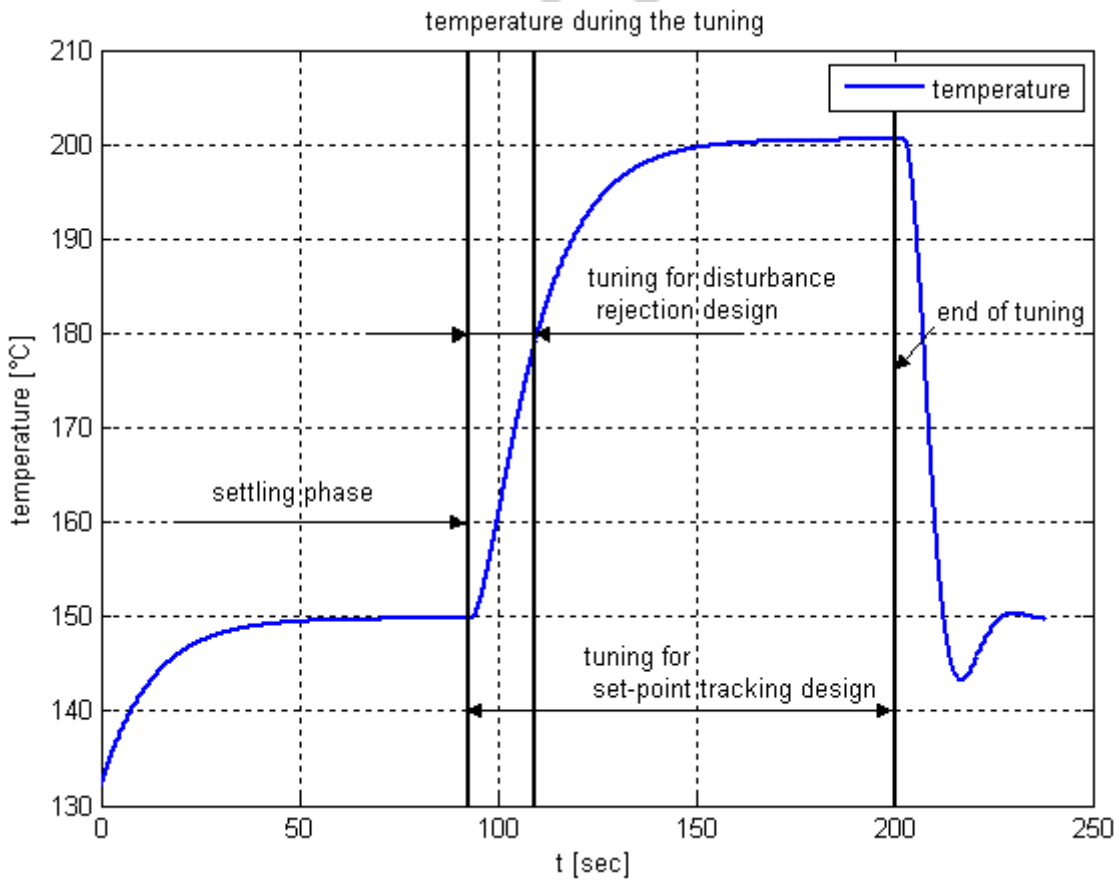


Fig. 8: Tuning for reference variable design SlimPID()

Task: SlimPID() auto-tuning with step response

Use your existing project and carry out auto-tuning using step response.

Set the request input to LCRPID_TUNE_REQU_OSCILLATE (2) to execute a step response, A disturbance rejection design (non-periodic) is used to determine the PID parameters.

Now perform a tuning procedure that determines the PID parameters using the reference variable design (non-periodic) (*request* = 4112).

Note the PID parameters from both tuning procedures for comparison.

Solution approach:
First set the set temperature to an operating point, in which the required manipulated variable *Y0* is already known (e.g. 150 °C)!

Set the inputs *Y0 and Y1* of the internally-used function block LCRPIDTune() in such a way that the temperature on the system changes significantly when a manipulated variable jump occurs.
Y0 can be determined using the trace of the previous example. Set *Y0* to the manipulated variable, which the controller had when in steady state in the operating point. Choose *Y1* to be approximately 30% larger than *Y0*.

Perform both tuning procedures.
Record the set temperature, the actual temperature and the manipulated variable in Trace.

Now compare the control parameters that were determined during the reference variable / disturbance variable design.

5. CONTROLLING TEMPERATURE SYSTEMS

There are generally two opposing manipulated variables when controlling temperature systems; one for heating and one for cooling. In most cases, the two manipulated variables have a different gain. Therefore, the system must be controlled using two different PID parameter sets.

5.1 Function block LCRTempPID()

This function block is specially designed for controlling temperature systems and should only be used for this purpose. The necessary PID parameters can be transferred to the function block using a type *lcrtemp_set_typ* structure connected with the *pSettings* input. The LCRTempTune() function block can be used to determine the parameters if they are not known.

Task: Control a temperature system (heating and cooling) using LCRTempPID()

Use the function block LCRTempPID() to construct the following control loop:

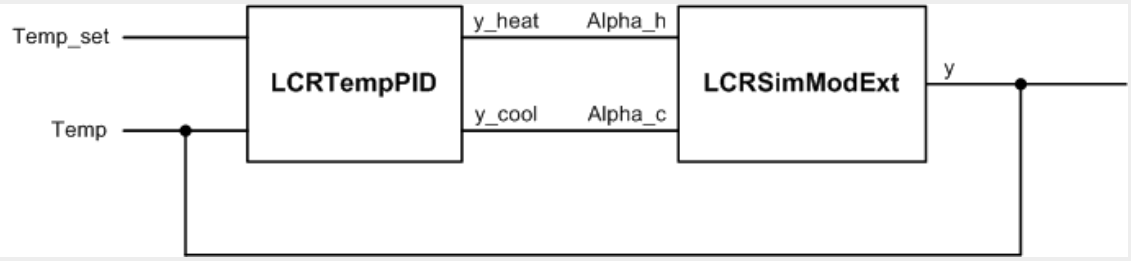


Fig. 9: Block diagram – control loop with LCRTempPID()

Use the online help to get more information about the LCRTempPID() function block, how it's used, and its operation.

Configure the controller function block using the type *lcrtemp_set_typ* structure connected to the *pSettings* input. There, the PID parameters must be placed in the substructure *PIDpara*.
You can refer to the examples in the online help to find suitable settings for the controller.

Perform set value jumps. Record the set temperature, actual temperature, the manipulated variable for heating (*y_heat*) and the manipulated variable for cooling (*y_cool*) in a trace.

Solution approach:

Parameters are automatically replaced with the default values internally if you do not specify them in the structure *lcrtemp_set_typ*. The default values are specially intended for extruders.
The PID parameters have to be configured.

The dynamics of the closed control loop can be influenced by the factors *dynGen, dynHeat and dynCool*, in particular with LCRTempPID(). *Kp_h* should be reduced if excessive oscillations occur. You can increase *Tn_h* if the control loop does not stabilize quickly enough.

5.2 Function block LCRTempTune()

The function block LCRTempTune() provides a procedure for automatically determining suitable control parameters, specially optimized for temperature systems. After the tuning procedure is complete, the function block writes the determined PID parameters into the type *lcrtemp_set_typ* structure connected with the pSettings input.

Principle tuning procedure:

- Settling phase before the heating procedure:
  The actual temperature must be close to the ambient temperature and the temperature changes are not allowed to be too great.
- Heating procedure:
  100% heating manipulated variable is used for heating until the temperature is close to the set temperature.
- Settling phase before the cooling procedure:
  The actual temperature is regulated to the set value in this phase. The cooling procedure is started after the transient effect is finished.
- Cooling procedure:
  100% cooling manipulated variable is used for cooling until suitable parameters have been found. Tuning is then successfully completed.

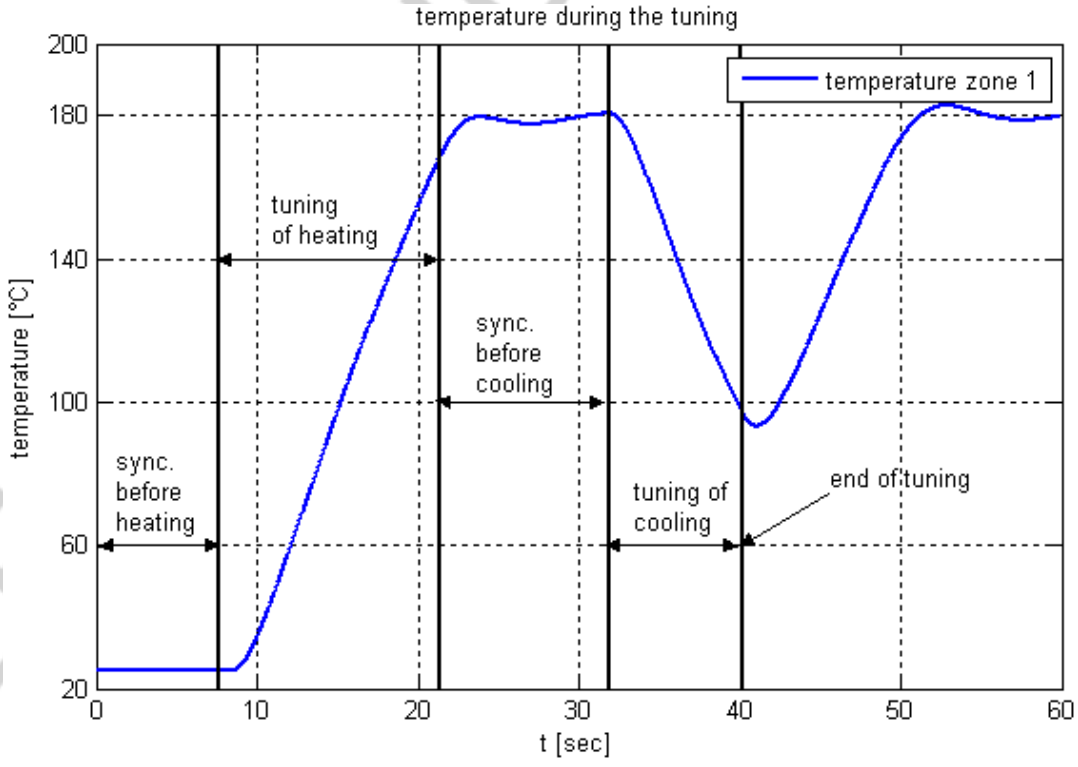


Fig. 10: Standard tuning temperature curve LCRTempTune()

Task: Tuning a temperature system using LCRTempTune()

Use the function block LCRTempTune() to construct the following control loop:

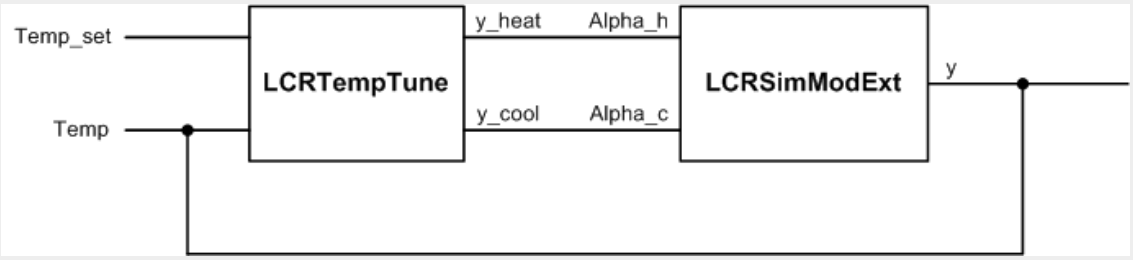


Fig. 11: Block diagram – control loop with LCRTempTune()

Use the online help to get more information about the LCRTempTune() function block, how it's used, and its operation.

Configure the function block using the structure (*lcrtemp_set_typ*) connected with the *pSettings* input. The tuning options can be entered to the TuneSet substructure. Suitable settings can be found in the online help.

Perform a tuning procedure in standard mode. Record the set temperature, actual temperature, the manipulated variable for heating (*y_heat*) and the manipulated variable for cooling (*y_cool*) in a trace.

LCRTempTune() writes the PID parameters determined during the tuning procedure to the structure connected with the *pSettings* input.

Solution approach:

Connect the *rdyTo* outputs to the *okTo* inputs. For example, the *rdyToHeat* output indicates when the system is ready for starting the heating procedure (*rdyToHeat* = TRUE). However, heating is not started until *okToHeat* is set to TRUE. In this case, connect the *rdyToHeat* output to the *okToHeat* input. The *rdyToCool* and *rdyToCoolEnd* must also be linked in the same manner.

Since this example deals with a simulation, a few tuning settings must be changed because the default values for are optimized for systems relevant to actual use (e.g. extruder).

If the LCRTempTune() function block is not disabled after the tuning procedure is complete, then the system will be temporarily regulated with the help of an integrated PID controller. However, the LCRTempPID() function block should be used to achieve optimum controller behavior (particularly with set value jumps).

5.3 Communication between LCRTempTune() and LCRTempPID()

Once the tuning procedure with the LCRTempTune() function block is complete, the determined PID parameters are automatically written to the structure connected with the *pSettings* input. This structure must also be connected with the *pSettings* input of the LCRTempPID() function block in order to enable communication between the two function blocks.

The following image illustrates which function block accesses which parameters in the communication structure and how (read, write, or both).

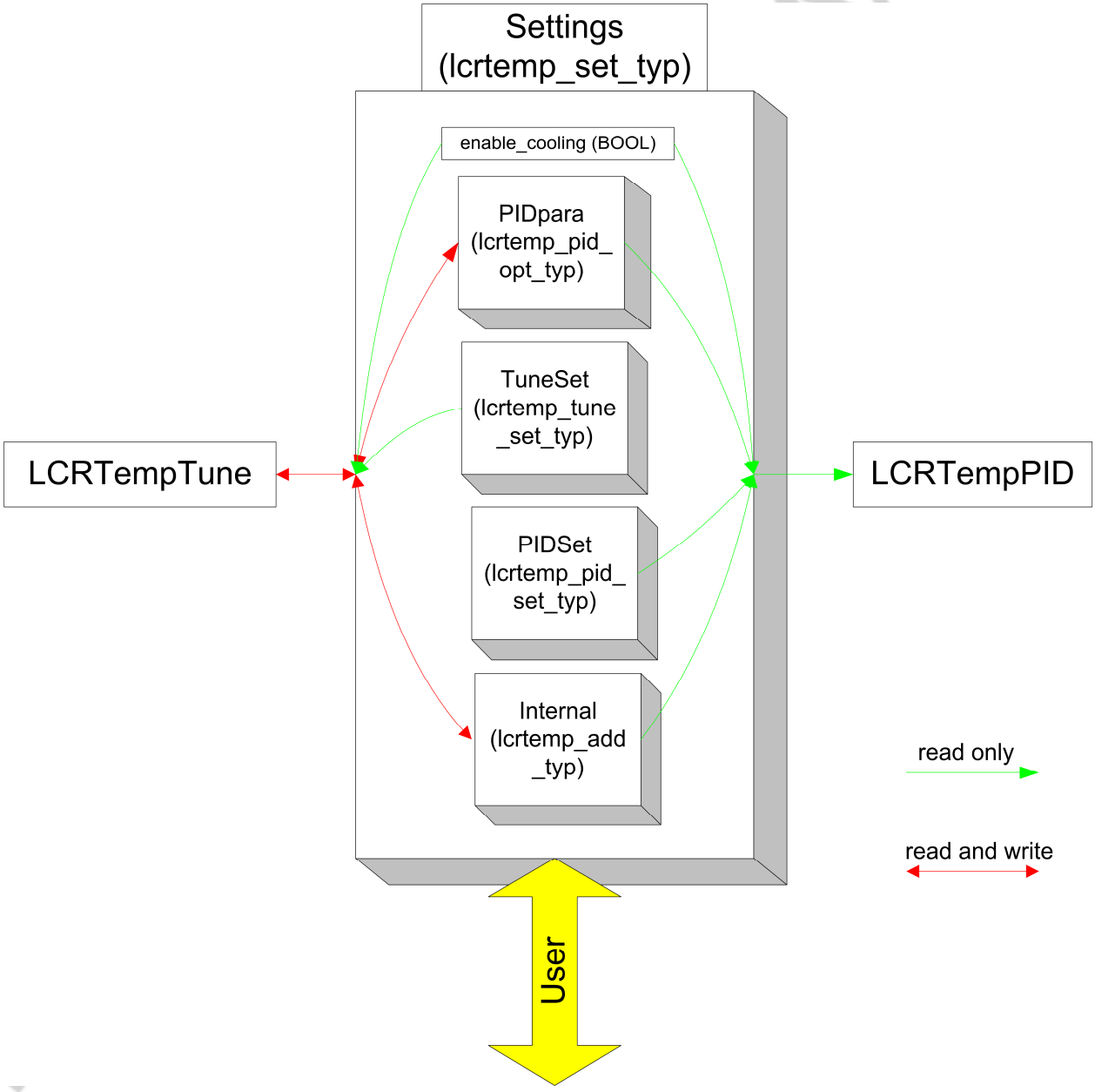


Fig. 12: Communication structure between LCRTempPID() and LCRTempTune()

5.4 Synchronized tuning of controlled systems

In practical application, multiple controlled systems must often be tuned simultaneously because they are placed next to one another and therefore affect each other. Each of these controlled systems contains its own actual value sensor and is controlled by a separate controller. The LCRTempTune() function block offers the possibility to synchronize multiple tuning procedures.

The *rdyTo* outputs of the LCRTempTune() function block are set to *TRUE* when the settling phases / tuning phases are complete.

The corresponding *okTo* inputs cannot be set simultaneously to *TRUE* on all LCRTempTune() function blocks until the *rdyTo outputs* = *TRUE* on every LCRTempTune() function block.

> **Note:**
>
> Examples about how to link the *rdyTo* output with the *okTo* inputs can be found in the online help by the function description of the LCRTempTune().

The following image shows an extruder with two adjacent heating and cooling zones that must be tuned synchronously due to their influence on each other.

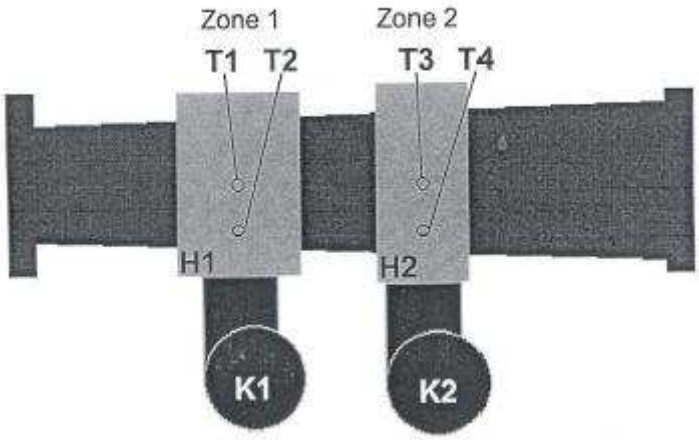


Fig. 13: Extruder model

Task: Synchronous tuning of two temperature systems using the function blocks LCRTempTune() and LCRTempPID()

Use the function blocks LCRTempPID() and LCRTempTune() to construct the following control loop twice:

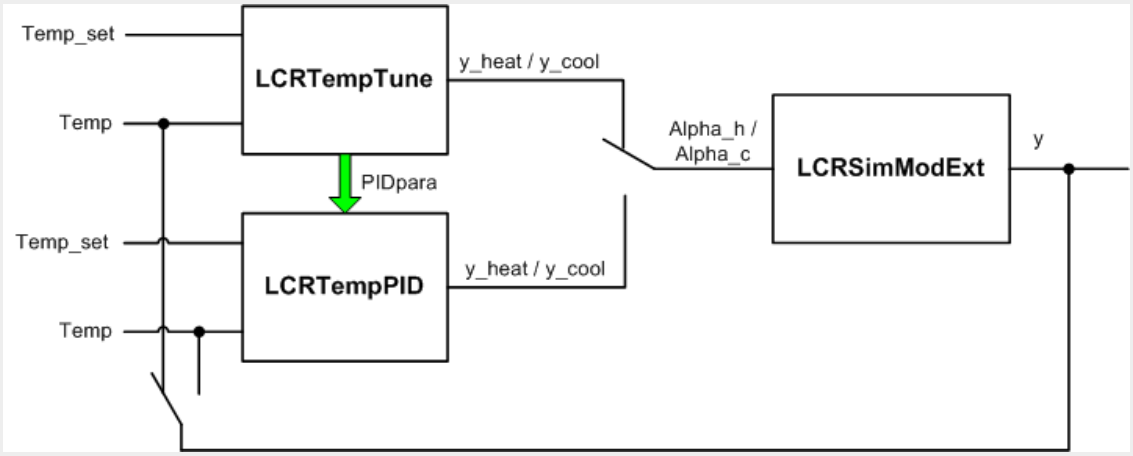


Fig. 14: Block diagram – control loop with LCRTempPID() and LCRTempTune()

First perform a tuning procedure in standard mode. Make sure that both zones are tuned synchronously. Record the actual temperatures, the manipulated variables for heating, the manipulated variables for cooling and the *status* outputs of the LCRTempTune() function blocks using trace.

Once the tuning procedure is complete (*done* = TRUE), switch off LCRTempTune() and enable the LCRTempPID(). Also do not forget to switch the manipulated variable outputs (*y_heat and y_cool*), which affect the system.

Solution approach:

Create a manipulated variable for heating / cooling each zone. This variable is written by LCRTempTune() during the tuning procedure. The manipulated variable is written by LCRTempPID() once the tuning procedure it complete.

Recording the *status* can help determine whether any warnings, which could have affected the tuning results, were output during the tuning (e.g. turning point not detected). If this occurs, the tuning options should be adjusted in the *TuneSet* structure.

6. IMPLEMENTATION OF A PULSE WIDTH MODULATION

The LCRPWM() function block can be used to implement a pulse width or pulse frequency modulator. This function block transforms an analog input signal into a digital, pulsed output signal. The input signal *x* is limited by *max_value* and *min_value*. The *t_min_puls* input can be used to specify the minimum duty cycle in seconds. A value larger than *t_min_puls* must be specified for the period duration *t_period*.

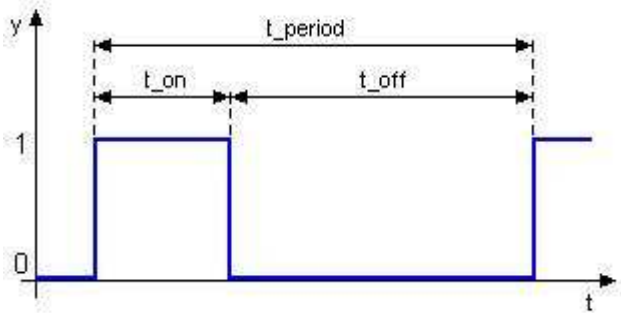


Fig. 15: Pulse width modulation

A pulse with the specified duration *t_min_pulse* is output, and the period is simultaneously extended if an input signal is present, which creates a pulse duration shorter than the minimum pulse duration (*t_min_pulse*). The period is extended in such a manner that the ratio from the switch-on duration to the switch-off duration is always equal to the input signal.

In the event that an input signal is specified which generates an idle time shorter than *t_min_pulse*, then a pulse pause with the duration of *t_min_pulse* is output and the period is extended to reach the correct pulse/pause ratio.

7. B&R SIMULATION MODEL 4SIM.00-01

Now let's have a look at a real temperature controlled system. To do this, we will be using the B&R simulation model with such a system to be controlled already integrated.

The temperature controlled system consists of a heating transistor, which heats up a heat sink. A fan is attached to the end of the heat sink. The temperature is measured using a PT1000 sensor.

The heating transistor is adjusted by applying voltage to the X2-14 pin, thereby causing the heat sink to heat up. A voltage of 24VDC or 10VDC must be selected and applied if the transistor is adjusted to full capacity (full thermal output). An analog output on a PLC is required (0-10VDC) if the transistor is to be adjusted in an infinitely variable manner. It is also possible to adjust the transistor via PWM (24VDC).

The fan is driven via the X2-15 pin and works according to the same principle.

Overtemperature protection is activated automatically if overtemperature occurs on the transistor (approximately 60°C). It remains in place until the temperature sinks sufficiently. If overtemperature protection is active, an LED labeled "TEMP" lights up on the front side of the model. A 24 VDC level is also set to LOW on the X2-18 pin. This can be evaluated using a digital input.

Fig. 16: B&R simulation model 4SIM.00-01

**Task: Pulse width modulation,
B&R simulation model 4SIM.00-01**

Use the LCRTempTune() und LCRTempPID() function blocks to regulate the temperature of the B&R simulation model.
(As an alternative to the B&R simulation model, you can also continue to use the LCRSimModExt() function block.)

Using this temperature control system, perform an auto-tuning.

First use an analog control with 0-10VDC, and then a digital control, so that you become comfortable with the function block LCRPWM().

Again, record the set and actual temperatures and the manipulated variables and analyze the resulting parameters.

Solution approach:

Unlike the previous task, you only have to build the control loop once and replace the LCRSimModExt() function block with the B&R simulation model. The actual value is read using an analog input; both control actions are output via two analog outputs.

You also have to make sure that the value ranges for the analog inputs and outputs are different from the value ranges of the function blocks. The following example illustrates:

	Analog temperature input	**Controller inputs (W, X)**
Data type	INT	REAL
Device	1 / 10 °C	1 °C

The input value must be converted from data type INT to REAL; the resolution must be converted from 1/10 °C to 1 °C.

	Controller outputs (Y1, Y2)	Analog outputs
Data type	REAL	INT
Value range	0 .. 100%	0 .. 32767

The function block outputs must be scaled from the value range 0 - 100% to 0 - 32767 and converted from data type REAL to INT.

You can also use the analog outputs for digital control with pulse width modulation as well. The SEL() function can be used for converting the digital signal into analog. The conversion takes place as follows:

	Pulse width modulation	Analog outputs
Data type	BOOL	INT
Value range	0, 1	0, 32767

The exact configuration of the LCRPWM() function block is explained in the online help. The length of the period *t_period* of the pulse width modulation can be 1.0 s for the heating control action and 10.0 s for the cooling control action. About 1/10 of the period is usually used as the minimum pulse length *t_min_pulse*.

Set the set temperature to 50 degrees so that an overshoot doesn't exceed the maximum temperature and activate the overtemperature protection.

Notes

Notes

PART II

Compendium
and
Reference work

(A theory-oriented approach)

8. DYNAMIC SYSTEMS

8.1 Motivation and definition

Closed loop control deals with influencing objects in a specific, targeted manner. The object (also known as the **controlled system, system** or **process**) is influenced in such a specific, targeted manner as to produce a desired behavior.

The variables that affect the system are called **input variables**. Input variables that are used to influence the system are known as **manipulated variables** or **control actions**; input variables that are beyond our control are known as **disturbance variables**. The system behavior can be approached via the output variables of the monitoring process. Output variables that are determined through measurement are known as **measurement variables**.

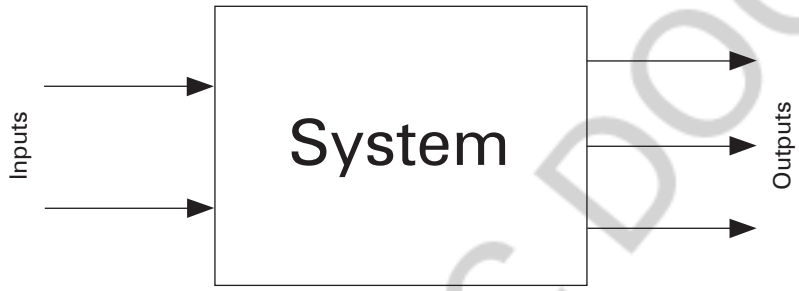


Fig. 17 Dynamic system

A system is considered **dynamic** if the output variables depend not only on the current value of the input variables, but also on their past. If the output variables depend only on the current value of the input variables, then the system is considered to be static.

In accordance with this definition, controlled systems as well as controllers (with integral element) and even control loops are generally dynamic systems.

Influencing a dynamic system in a targeted manner demands a certain degree of knowledge about its dynamic behavior. Dynamic systems can be described using **mathematical models** (model equations).

As the name itself implies, models are only approximated model notions of reality. It is the art of the engineer to create the simplest possible yet sufficiently accurate model containing the relevant properties of the real system.

The more accurate information you have about the behavior of a real controlled system (i.e. the more precisely the mathematical model corresponds to the real behavior of the system to be controlled), the more

accurately you can impress a desired behavior upon that controlled system (can this be controlled).

Differential equations are an important part of mathematical models in technical systems because several physical laws of nature are formulated mathematically using differential equations (e.g. in mechanics, thermodynamics and electrical engineering).

8.2 A mechanical example

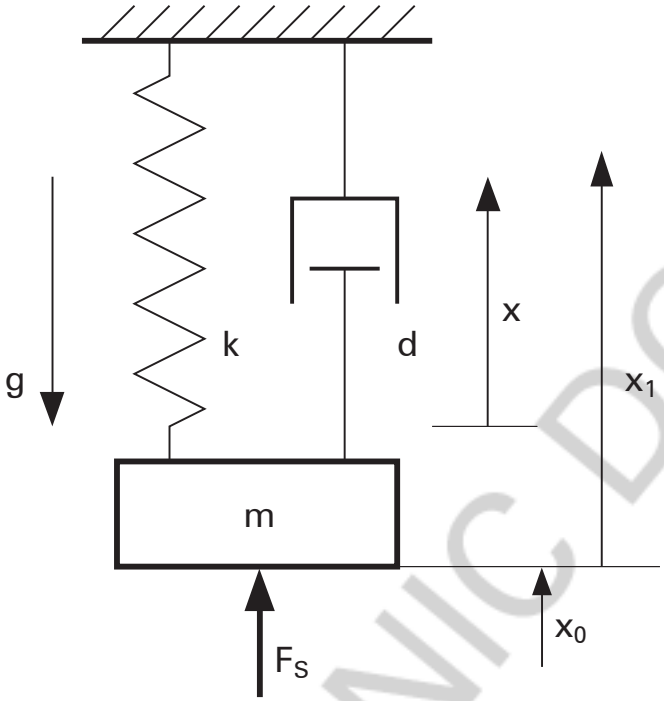


Fig. 18 Spring and mass system

A body with the mass m [kg] is hanging from a spring with the spring constant k [N/m], i.e. the following formula applies for the action of force on the body:

$$F_F = -kx,$$

That means that the spring is relaxed at the position $x = 0$. Movement of the body is decelerated by a speed-proportional attenuator with the attenuation constant d [Ns/m], i.e. the following formula applies for the action of force on the body:

$$F_D = -dv,$$

whereby $v = \dot{x}$ is the speed of the body. Additionally, the following gravitational force:

$$F_G = mg$$

acts on the body with the gravitational acceleration g [m/s^2] and a positioning force F_s [N].

The movement of the body is only possible in x-direction. Other actions of force (e.g. air friction, etc) are neglected.

The principle of linear momentum (second Newtonian axiom) sets the relationship between the resulting acceleration and the sum of the active forces:

$$m\ddot{x} = F_F + F_D - F_G + F_S$$

When written as a system of first-order differential equations:

$$\dot{x} = v$$
$$\dot{v} = \frac{1}{m}[-kx - dv - gm + F_S] \qquad \text{Eq. 1}$$

If there is interest in the position of the body, then it is selected as system output variable:

$$y = x$$

8.3 A thermal example: Extruder zone

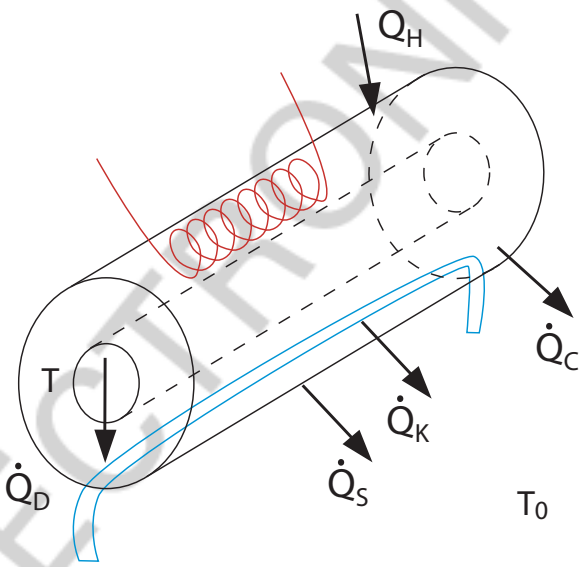


Fig. 19 Extruder zone

A metal block with melted plastic flowing through it (mass m [kg], specific thermal capacity c [J kg^{-1} K^{-1}], emissivity ε [1] and surface A [m^2]) is tempered by a heater with the thermal output $\dot{Q}_H$ [W] and a cooling unit with the cooling capacity $\dot{Q}_C$ [W].

The temperature at the center of the block T is measured using a temperature sensor. The extremely simplified assumption is made that the entire block has the homogeneous temperature T.

The heat transmission to the environment via convection and thermal conduction is:

$$\dot{Q}_K = \alpha A(T - T_0),$$

whereby α [W m^{-2} K^{-1}] is the heat transfer coefficient and T_0 [K] is the ambient temperature. The heat transmission to the environment via radiation is:

$$\dot{Q}_S = \varepsilon\sigma A(T^4 - T_0^4),$$

thereby σ [W m^{-2} K^{-4}] is the Stefan-Boltzmann radiation constant and ε [1] is the emission coefficient. The heat transmission from the melted plastic to the metal block is $\dot{Q}_D$ [W]. This value cannot be measured and therefore represents a classic disturbance variable for the controller.

The first law of thermodynamics sets the relationship between the dissipation of the body's internal energy and the sum of the acting thermal flows

$$\dot{E} = mc\dot{T} = \dot{Q}_H - \dot{Q}_C - \dot{Q}_K - \dot{Q}_S + \dot{Q}_D. \qquad \text{Eq. 2}$$

8.4 Characteristics of dynamic systems

8.4.1 Time invariance

A system is consider to have time invariance if temporal shifting of the input variables by the time span τ results only in a temporal shift of the output variables by the same time span τ.

The spring-and-mass system as well as the extruder zone are both time invariant systems. One way to achieve linear time invariant systems is to linearize non-linear systems along trajectories.

8.4.2 Linearity

The superposition principle applies to linear systems: The following equation:

$$y_1 = G \cdot u_1$$

is the system's response to the input signal u_1 and the following equation:

$$y_2 = G \cdot u_2$$

is the system's response to the input signal u_2. If

$$\alpha_1 y_1 + \beta_2 y_2 = G \cdot (\alpha_1 u_1 + \beta_2 u_2),$$

then the system is linear.

Linearity says that the system behaves the same at every operating point. In this case, a operating point is determined by a specific value of the controlled **and** manipulated variables. A system is exactly linear when it has the same transfer function at every operating point.

The model of the extruder zone from section 8.3 would be linear if the heat lost via radiation could be neglected relative to the heat lost via convection.

For the realistic numeric values $T = 200°C$, $T_0 = 25°C$, $\alpha = 8\ W\ m^{-2}\ K^{-1}$, $\varepsilon = 0.7$, and $\sigma = 5.67e\text{-}8\ W\ m^{-2}\ K^{-4}$, the following equation results for the ratio of heat loss:

$$\frac{\dot{Q}_S}{\dot{Q}_K} = \frac{\varepsilon\sigma(T^4 - T_0^4)}{\alpha(T - T_0)} = 1.2,$$

That means that in this case, there is a higher proportion of radiation and linear system behavior cannot be expected.

8.4.3 Single and multi variable systems

A system with just one input variable and just one output variable is considered a single variable system. A multivariable system has more than one input and output variable.

Temperature control for the extruder zone from section 8.3 is a multivariable system, even though it has just one controlled variable (output variable of the dynamic system), because there are two control actions (input variables). Each control action has a separate transfer function.

8.4.4 Stability

There are different definitions of stability. BIBO stability evaluates the system's transfer behavior: A system is BIBO stable (**B**ounded **I**nput **B**ounded **O**utput), if it responds to limited input variables with output signals that are also limited.

The thermal example in section 8.3 is BIBO stable because the heat transmission to the environment stabilizes the system to specific temperature levels for limited control actions.

The mechanical example from section 8.2 is BIBO stable, if the damping constant is $d > 0$. The damper converts kinetic energy to frictional energy (heat). The system is not BIBO stable if $d = 0$, because the output variable can increase beyond all variables when there is limited excitation with resonance frequency.

Excitation with the resonance frequency can destroy the system (resonance catastrophe) even for non-zero, but minor damping. The best known example of a resonance catastrophe is the collapse of the suspension bridge in Angers in the year 1850, triggered by 730 French soldiers marching lock-step across the bridge. 226 soldiers were killed in the incident. On the other hand, the collapse of the Tacoma Narrows Bridge (http://www.ketchum.org/bridgecollapse.html) was caused by aerodynamic-induced wobbling instability instead of forced resonance.

A positioning drive with the controlled variable position x (equal to the output variable of the dynamic system) and the manipulated variable drive torque M (equal to the input variable of the dynamic system) is not BIBO stable because the system represents the output variable x through doubled integration of the input variable M. This system's transfer function is (without considering frictional torques, etc.):

$$G(s) = \frac{x(s)}{M(s)} = \frac{R}{s^2 I},$$

whereby R is the radius subject to the torque and I is the drive's total moment of inertia.

Chemical chain reactions represent another example of non-BIBO-stable systems.

8.5 Description methods

8.5.1 Description in the time domain (state space)

In addition to the knowledge of the input variables $u_1, \ldots, u_m$ and the output variables $y_1, \ldots, y_l$, the state variables $x_1 \ldots x_n$ must also be known in order to describe a dynamic system in the time domain.

Together with the input variables, the state variables uniquely describe the curve of the output variables. The number of state variables n is referred to as the dimension (or order) of the system and is equal to the number of first-order differential equations required to describe the system.

The term "time domain" indicates that the state equations are differential equations in time and that all input, state and output variables are time functions (time signals).

The mechanical example from section 8.2 has two state variables (the position x and speed v of the body), one input variable (the positioning force F_S) and one output variable (the position x of the body).

The thermal example from section 8.3 has just one state variable (the temperature of the metal block T), two input variables (the thermal output $\dot{Q}_H$ and the cooling capacity $\dot{Q}_C$) and one output variable (the temperature of the metal block T).

A higher system order allows for more complex system behavior. In principle, a first-order system (e.g. a PT_1 element, e.g. single low pass) is not capable of oscillation. A second-order system could be capable of oscillation. A chaotic system is at least third-order and non-linear.

The description in the time domain is useful for finding stationary operating points in systems. These are found by zeroing the derivatives of the state variables:

The stationary operating points are taken from equation 1 for the mechanical example from section 8.2:

$$0 = v$$
$$0 = \frac{1}{m}\left[-kx - dv - gm + F_S\right]$$

for

$$x = \frac{F_S - gm}{k}. \qquad \text{Eq. 3}$$

Due to the gravitational force, the body will occupy the following position if there is no positioning force:

$$x_0 = -\frac{gm}{k}$$

Equation 3 can be used to calculate a feed-forward for the positioning force:

$$F_{VS} = gm + kx_{set}$$

for positioning to the position $x = x_{set}$.

Task: Calculate a feed-forward (under the premise that the radiation heat transmission to the environment can be neglected, $\dot{Q}_S = 0$), to temper the extruder zone (from section 8.3) through which no material is flowing ($\dot{Q}_D = 0$) to the temperature T_{set}. Which deviation from this set temperature results when this extruder zone is operated with this feed-forward without superposed control and when a heat transmission occurs from the material to the zone of $\dot{Q}_D$ during operation.

8.5.2 Description in the frequency domain (transfer behavior)

If all of the system equations (state differential equations and equations for the output variables) of a dynamic system are linear and time invariant, then these equations can be subjected to the Laplace transformation. Algebraic equations in the following new complex variable result from the differential system equations in time:

$$s = \alpha + j\omega,$$

whereby $\omega = 2\pi f$ can be used as angular frequency for the input or output signals. The quotient from the output signal and the input signal of a system:

$$G(s) = \frac{y(s)}{u(s)}$$

is known as the transfer function and describes, which frequency spectrum of the output signal y(s) the system can use to respond to the frequency spectrum of the input signal u(s).

The view of a system in the frequency domain is a view of the transfer behavior. It sets the relationship between frequency spectrums of input and output signals with each other.

The relationship between input and output variables is calculated from the following equation for the mechanical example in section 8.2:

$$\ddot{x} = \frac{1}{m}\left[-kx - d\dot{x} - gm + F_S\right]. \qquad \text{Eq. 4}$$

This equation is affine (and therefore non-linear!) in x and therefore cannot be subjected to Laplace transformation. The following variable transformation:

$$x = x_1 + x_0$$

with $x_0 = -\frac{gm}{k}$ (the new coordinate x_1 now starts at the stationary position of the mass without affecting the positioning force x_0), is calculated from equation 4, whereby $\dot{x}_1 = \dot{x}$ and $\ddot{x}_1 = \ddot{x}$:

$$\ddot{x}_1 = \frac{1}{m}\left[-kx_1 - d\dot{x}_1 + F_S\right].$$

This equation is linear. The Laplace transformation is:

$$ms^2 x_1(s) + dsx_1(s) + kx_1(s) = F_S(s).$$

The transfer function from the input variable positioning force to the output variable position x_1 is a PT_2 element of the form:

$$G(s) = \frac{x_1(s)}{F_S(s)} = \frac{1}{ms^2 + ds + k}. \qquad \text{Eq. 5}$$

A transfer function can be represented in a Bode diagram. The magnitude characteristic in the Bode diagram indicates (in [dB]) how the frequencies contained in the system's input signal are amplified and weakened. The phase characteristic of the Bode diagram indicates (in [°]) which phase shift in the frequencies contained in the input signal pass through the system.

Fig. 15 shows the Bode diagram of the transfer function equation 5 for the parameters $m = 1$ kg, $d = 2$ Ns/m, $k = 10$ N/m.

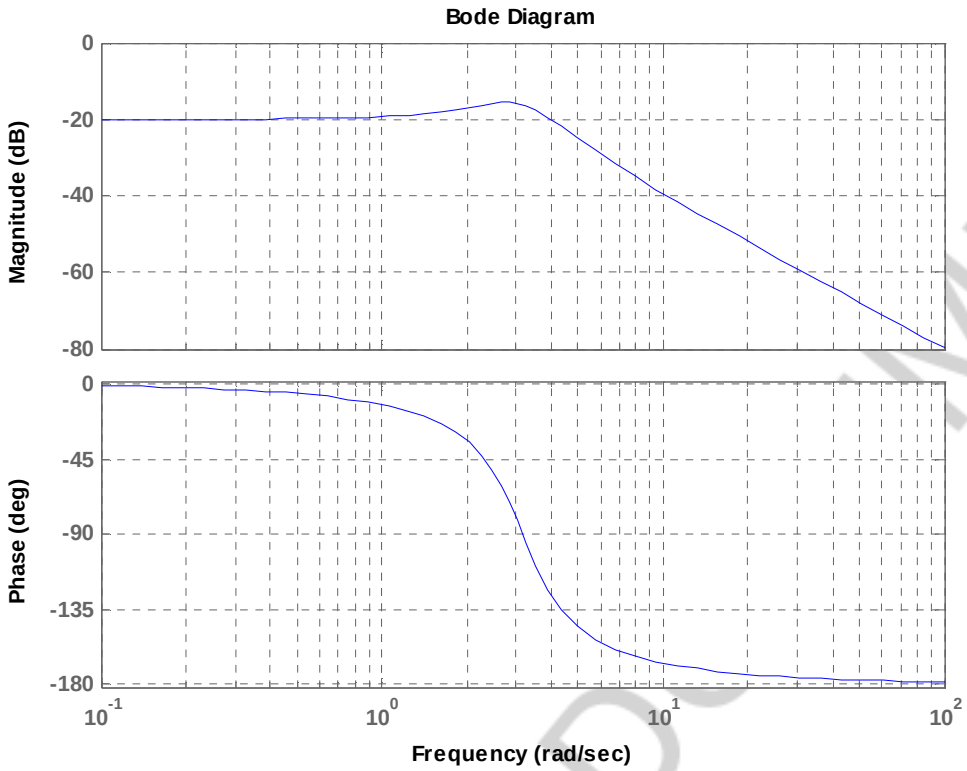


Fig. 20 Bode diagram of the transfer function $G(s) = (s^2+2s+10)^{-1}$

For the detailed analysis (oscillation capability, natural frequency, resonant rise, etc.) of a second-order delay element (PT_2), please refer to the academic literature (e.g. W. Haager: Regelungstechnik, ISBN 3-209-00928-7 – in German).

9. CONTROLLED SYSTEMS

As discussed already in section 8, a system which must be controlled can be more accurately controlled when you know more about its behavior. Mathematical models are used to describe a system which must be controlled (e.g. transfer functions).

A mathematical model can be created by establishing a theoretical model and/or carrying out experimental identification.

9.1 Establishing a model

When establishing a theoretical model, the mathematical model of the system to be controlled is derived from the basic laws of physics (see examples in section 8). This produces detailed information about the system:

- Basic type of system behavior
- Influence of all system parameters on its behavior

If some of the system parameters are unknown (which is often the case), then an inference can be made based on the basic type of system behavior, but the coefficients in the transfer function cannot be calculated.

For the extruder zone from section 8.3, neglecting the heat transmission to the environment (via radiation) with the temperature difference compared to the environment, :

$$T_1 = T - T_0$$

results in the transfer function of the heating control action (PT_1 element):

$$G(s) = \frac{T_1(s)}{\dot{Q}_H(s)} = \frac{k_S}{1 + T_G s} = \frac{\frac{1}{\alpha \cdot A}}{1 + \frac{m \cdot c}{\alpha \cdot A} s}.$$

The system gain:

$$k_S = \frac{1}{\alpha \cdot A}$$

is reduced, the greater the heat transmission coefficient and the surface of the zone. The time constant:

$$T_G = \frac{m \cdot c}{\alpha \cdot A} = m \cdot c \cdot k_S$$

is larger, the greater the thermal capacity $m \cdot c$ of the block and is linearly proportional to the system gain.

9.2 Identification

When dealing with complex controlled systems, a theoretical model cannot be established or cannot be determined within a reasonable amount of time. A model of the system to be controlled can then be determined using an experimental approach with identification. To do this, the system is excited with specific input signals and the reaction of the system is measured from which the system's transfer behavior is then concluded.

Fig. 21 shows the response of a real extruder zone to a jump in the heating manipulated variable from 30.5% to 61.0% at the time point $t = 2000\ s$. The steps response of a PT_1 element is shown with identical gain and rise time for comparison.

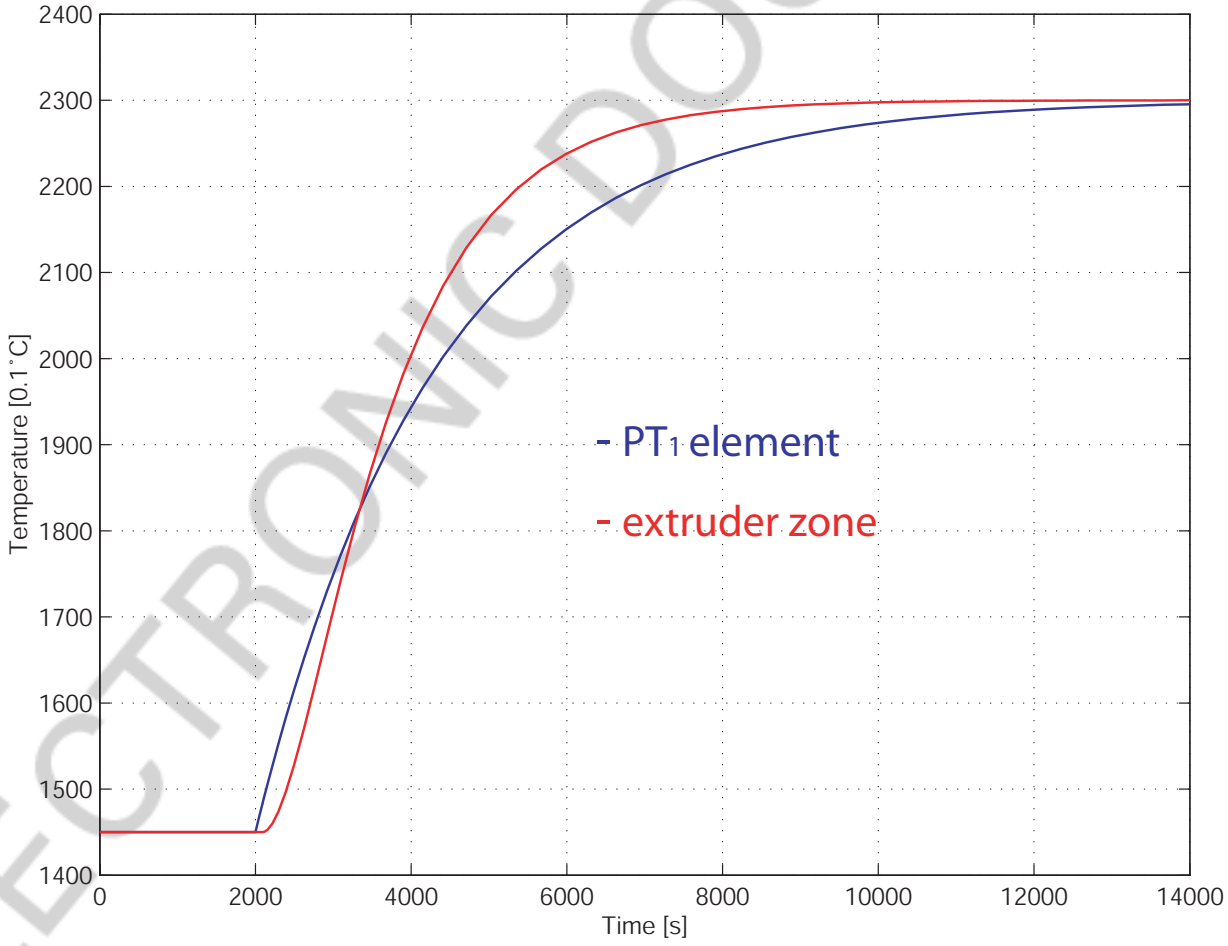


Fig. 21 System identification with step response

It can be seen clearly that the real system behavior deviates considerably from the PT_1 behavior (determined by establishing a theoretical model). The discrepancy is due to the extremely simplified model assumption that the entire block has the homogeneous temperature T. The thermal

conduction in the metal block results in a dead time (thermal waves have a finite velocity of propagation) and a higher-order delay in the real system behavior.

9.3 An important type of controlled system

Many industrial controlled systems have non-periodic (non-oscillation-capable) higher-order delay behavior (in some cases with additional dead time). The transfer behavior of such systems can be approximated (Fig. 22) using a first-order low pass with dead time:

$$G(s) = \frac{k_S}{1+T_G s} \cdot e^{-sT_U}$$

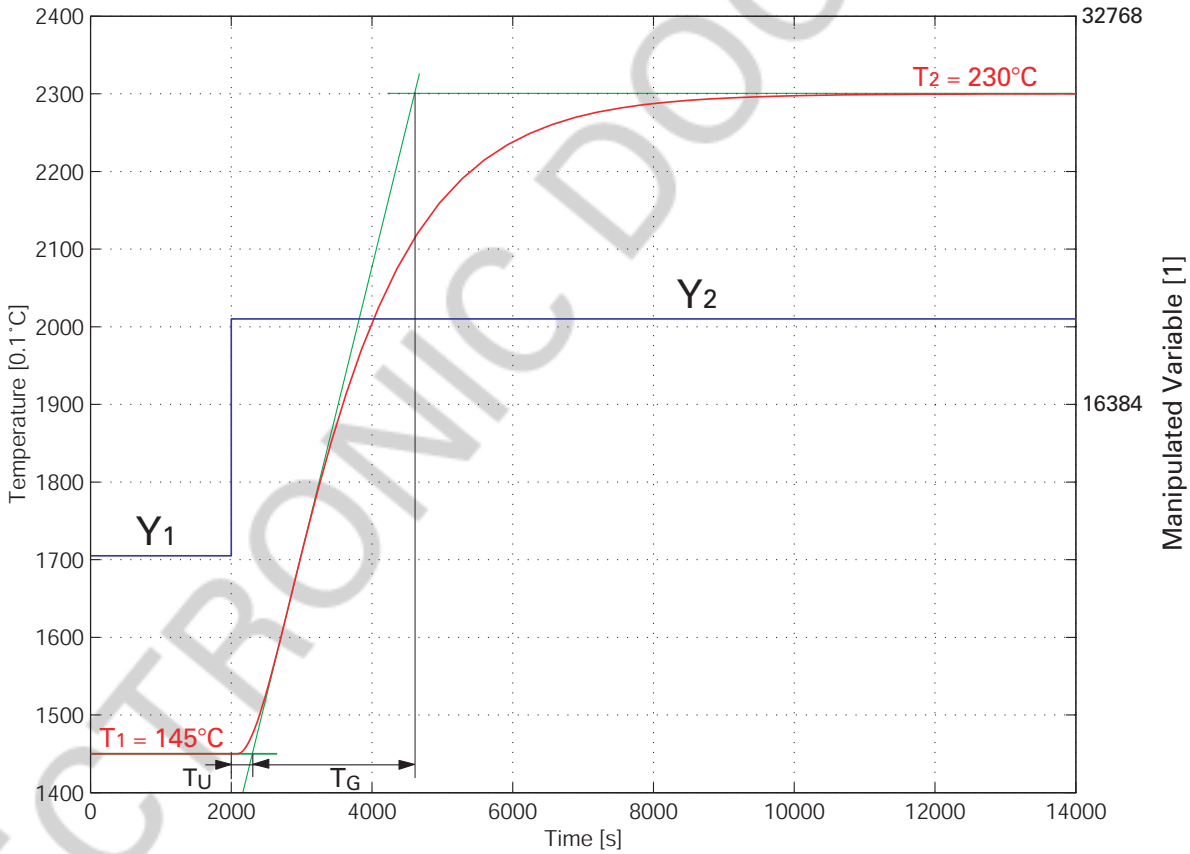


Fig. 22 Approximation of a system using a first-order low pass with dead time

The manipulated variable Y_1 is first connected to the system to be controlled (in this case $Y_1 = 10000$), which the system maintains at the desired operating point T_1 (or close to that point). If changes to the controlled variable can no longer be detected, then a manipulated variable jump:

$$Y_2 = Y_1 + \Delta Y$$

is applied to the system to be controlled (in this case $\Delta Y = 10000$) and the characteristic of the controller variable (process reaction curve) is recorded until once again no changes to the controlled variable can be detected and a new stationary operating point T_2 has been set.

The dead time (dwell time) T_U and the rise time T_G are determined from the intersection of the reversal tangent and the value of the controlled variable before the step / after the step. The system gain is calculated as follows:

$$k_S = \frac{T_2 - T_1}{Y_2 - Y_1} = \frac{\Delta T}{\Delta Y}.$$

In the event that the step response cannot be recorded until a final stationary value T_2 has been reached for the controlled variable (i.e. the stepping attempt is prematurely aborted), then neither the rise time nor the system gain can be determined. If the step response is aborted after reaching the inflection point, then the maximum slope $\dot{T}_{\max}$ of the controlled variable in the inflection point can be determined.

The system to be controlled is calculated as follows for the step response from Fig. 17:

$$G(s) = \frac{0.085}{(1+1250 \cdot s) \cdot (1+500 \cdot s)} \cdot e^{-90s}$$

when using this method, which results in an approximation with:

$$k_S = 0.085 \quad [0.1°C/1]$$
$$T_G = 2300 \quad [s]$$
$$T_U = 305 \quad [s]$$
$$\dot{T}_{\max} = 0.37 \quad [0.1°C/s]$$

There are simple formulas used to make the settings for controllers from the PID family (section 11.2) for these types of system models (PT_1T_T behavior).

10. THE CLOSED CONTROL LOOP

10.1 The basic principle of closed loop controllers

Controlling via closed loop control is automatically influencing a technical process (plant, system or controlled system) in a specific, targeted manner. Unlike **open loop controlling**, this is a closed loop, which means that the variables (manipulated variables) that influence the process are independently established with suitable control mechanisms (actuators) from measured process variables instead of being specified only externally. Closed loop control deals with the (mathematical) description of such control processes and the targeted design of closed loop controllers in such a manner so that these control process can be carried out as desired. The **basic principle of every closed loop controller** is the negative **feedback** (inverse feedback) from the variable that must be controlled.

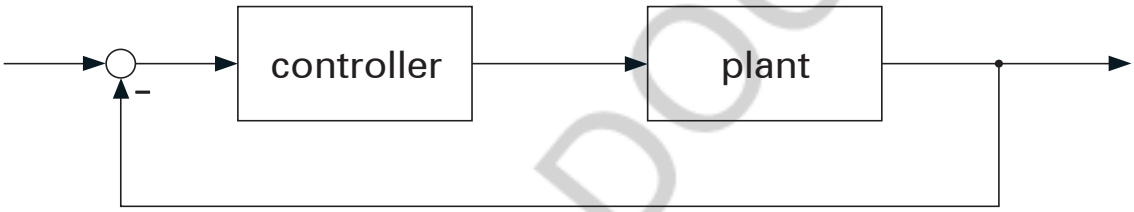


Fig. 23 Closed loop control

Fig. 24 Open loop control

10.2 Block diagram

A block diagram is the representation of a technical system (e.g. a control loop) using function blocks. The function blocks are connected to each other via defined inputs and outputs.

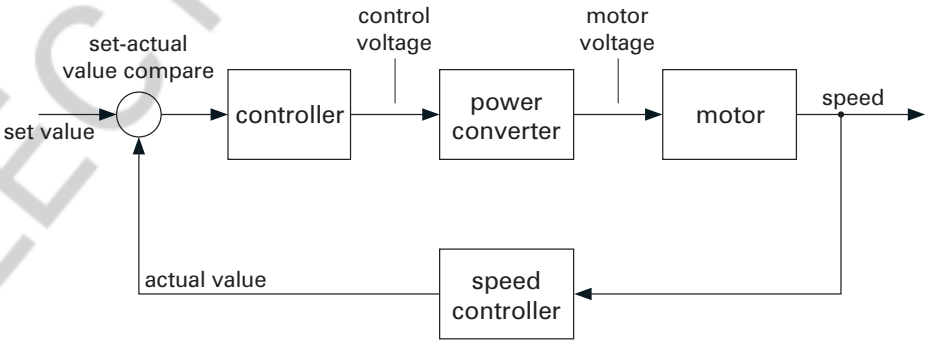


Fig. 25 Block diagram

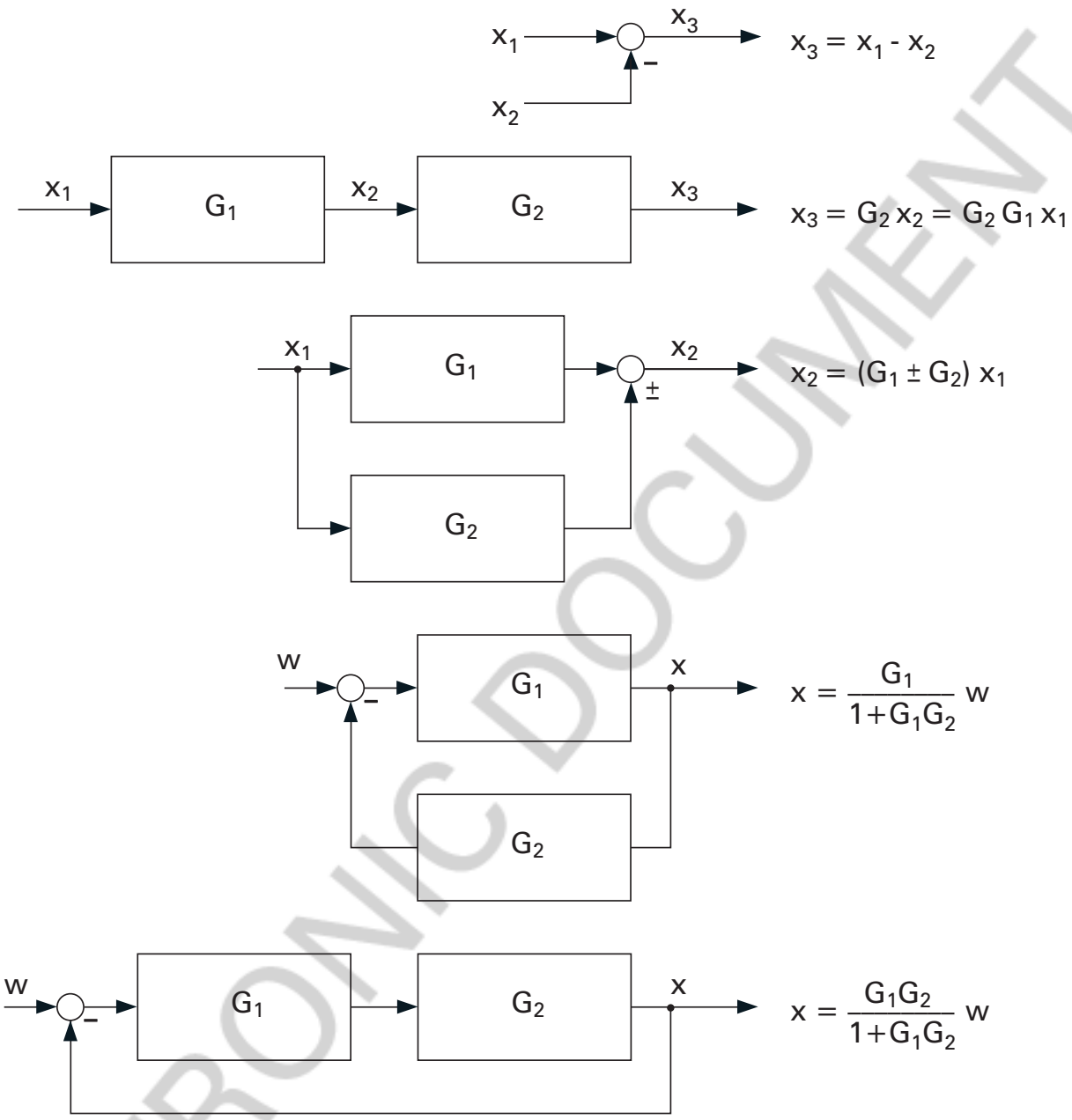


Fig. 26 Calculating with block diagrams

10.3 The standard control loop

Fig. 27 shows the block diagram for a standard control loop with the following elements:

- Controlled system: the system to be controlled (process or system).
- Controlled variable: the variable to be intentionally influenced by the controller (output variable of the controlled system or actual value)
- Reference variable: set value of the controlled variable (e.g. specified by operator)

- Measuring element (sensor): provides the controller with a measurement value of the controlled variable (typically via an input module)
- Control deviation: difference between a reference and controlled variable (between set and actual value)
- Controller: uses the control deviation to establish a corresponding signal, to affect the controlled system (typically via an output module)
- Actuator: the connecting element between the controller, which generally only provides weak signals, and the system to be controlled, which usually requires strong signals to have an effective influence. The output variable of the actuator is the manipulated variable.
- Disturbance variable: describes the influence of non-measurable variables, which affect the control loop

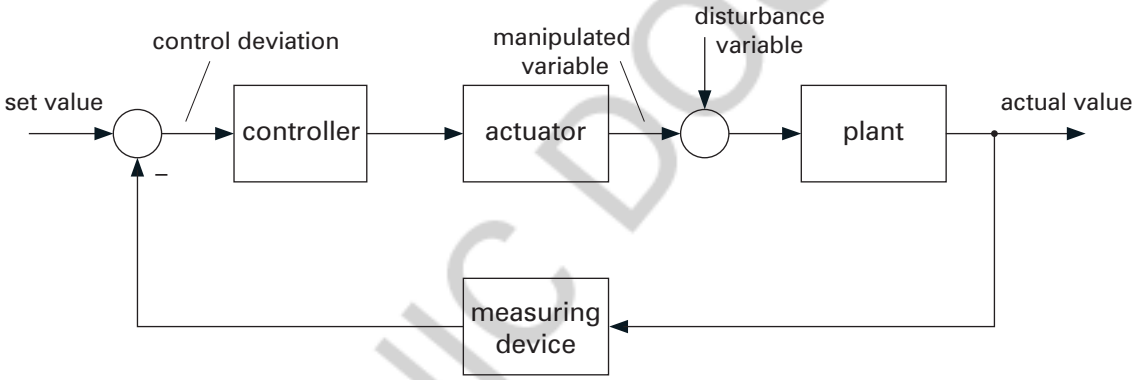


Fig. 27 Standard control loop

	Description
x	Controlled variable (actual value)
w	Reference variable (set value)
e	Control deviation e = w – x
y	Manipulated variable
z	Disturbance variable
R(s)	Controller transfer function
G(s)	Transfer function for the system to be controlled

The measuring element and actuator are generally assigned to the controlled system and drawn as a single block. As a result, the controlled variable is not the actual physical variable that must be controlled, e.g. pressure (in Pascal or bar) in a pressure controller, rather it is the corresponding measurement signal from the input module (e.g. as integer with a value range 0 - 32767). Likewise, this means that the controller

output is the same as the manipulated variable (e.g. as integer with a value range 0 - 32767). The resulting block diagram for the control loop is shown in Fig. 28.

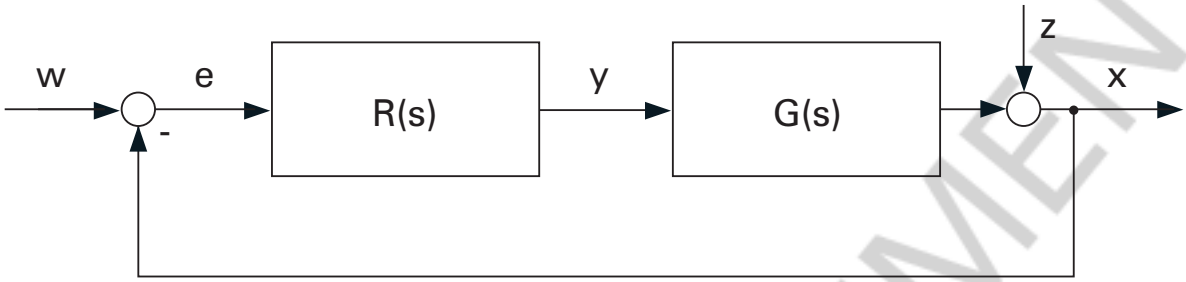


Fig. 28 Standard control loop II

10.3.1 Transfer function of the open loop

The transfer function of the open loop (without feedback) is:

$$L(s) = R(s) \cdot G(s) .$$

10.3.2 Reference transfer function

The transfer function of the closed control loop (reference transfer function) is:

$$T(s) = \frac{L(s)}{1 + L(s)} = \frac{R(s) \cdot G(s)}{1 + R(s) \cdot G(s)} .$$

A 'reference (variable) design' involves impressing a desired reference transfer function on a control loop. Ideally, $T(s) = 1$. However, this cannot be achieved due to the low pass character of real systems.

10.3.3 Disturbance variable transfer function

The disturbance variable transfer function is calculated as follows:

$$T_d(s) = \frac{1}{1 + R(s) \cdot G(s)} .$$

Ideally, $T_d(s) = 0$. However, this cannot be achieved either. A 'disturbance (variable) design' involves minimizing the disturbance variable transfer function.

When comparing $T(s)$ and $T_d(s)$, it becomes evident that a **controller design in the standard control loop is always a compromise between a set-point tracking design and a disturbance rejection design** because both transfer functions are determined by selecting a controller $R(s)$. In principle, a disturbance rejection design provides a large number of more dynamic

controllers, which compensate for disturbances better, but also exhibit considerable overshoot when reference variable jumps occur.

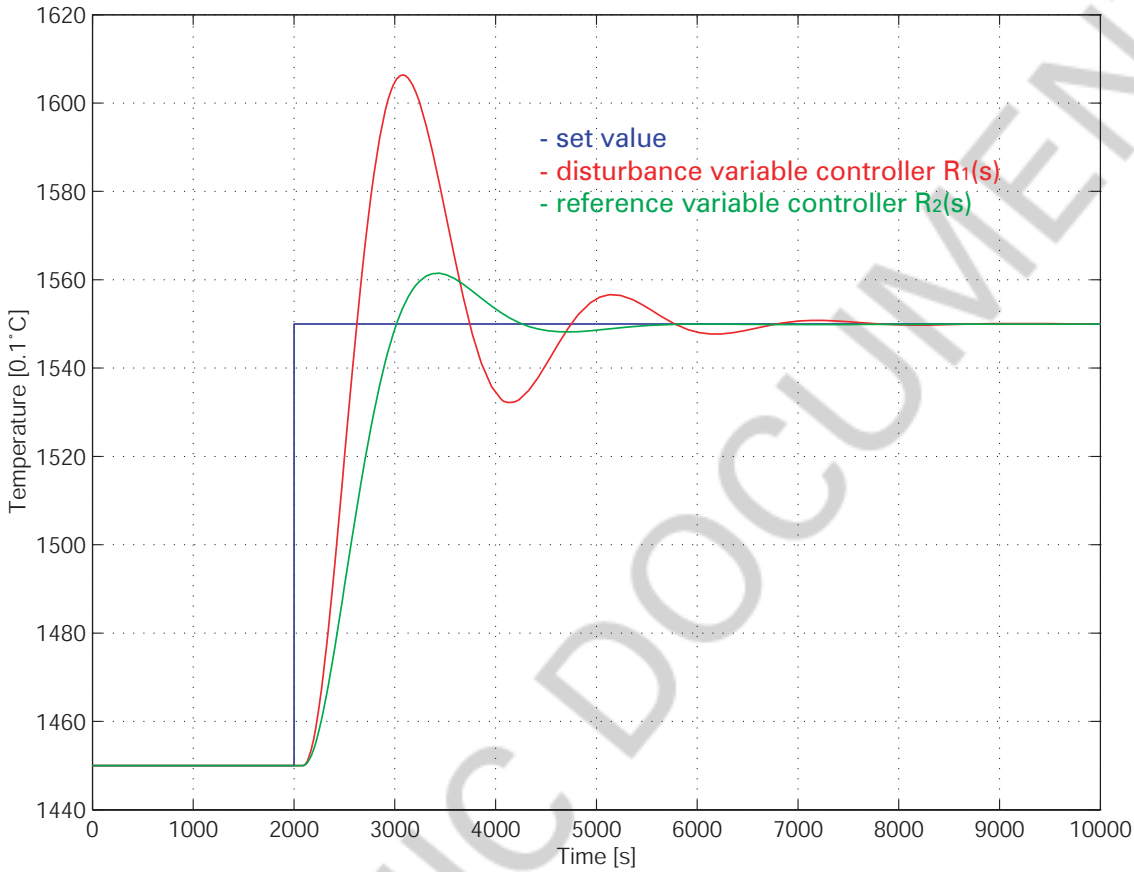


Fig. 29 Comparison of the set-point tracking behavior of a set-point controller and a disturbance rejection controller in the standard control loop

Fig. 29 shows a comparison of the responses to a reference variable jump in a standard control loop with the system to be controlled:

$$G(s) = \frac{0.085}{(1+1250 \cdot s) \cdot (1+500 \cdot s)} \cdot e^{-90s},$$

and a disturbance rejection controller:

$$R_1(s) = 88.61 \cdot \left(1 + \frac{1}{732 \cdot s} + 128 \cdot s\right)$$

and a set-point tracking controller:

$$R_2(s) = 53.17 \cdot \left(1 + \frac{1}{2300 \cdot s} + 153 \cdot s\right).$$

As a comparison, Fig. 30 shows responses from the same control loop to a disturbance variable jump.

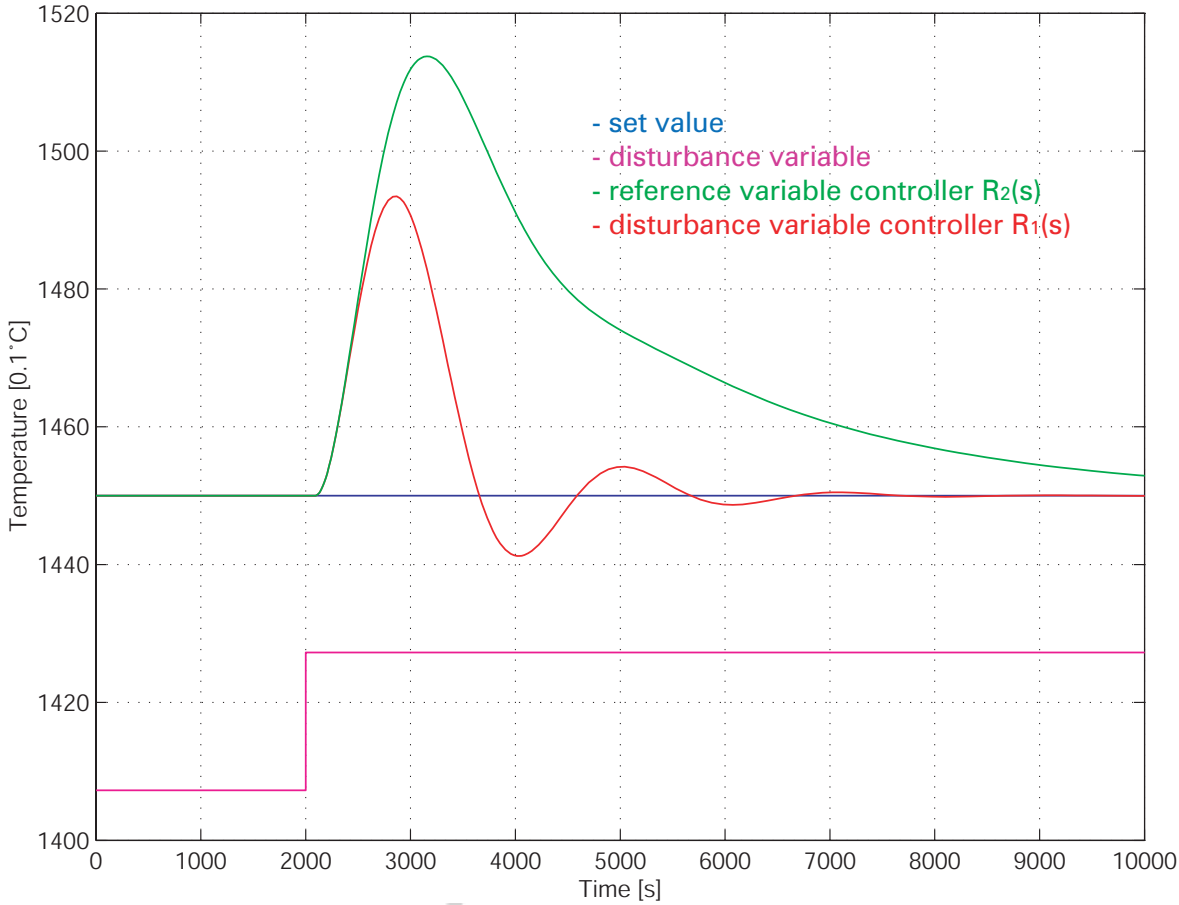


Fig. 30 Comparison of the disturbance rejection behavior of a set-point tracking controller and disturbance rejection controller in the standard control loop

In section 12.4, the standard control loop will be expanded in such a manner so that the reference and disturbance variable transfer functions (within certain limits) can be influenced separately.

10.4 Characteristics of closed control loops

10.4.1 Characteristics in the time domain

Simple characteristics from the step response are used to evaluate the quality of a set-point tracking controller's timing:

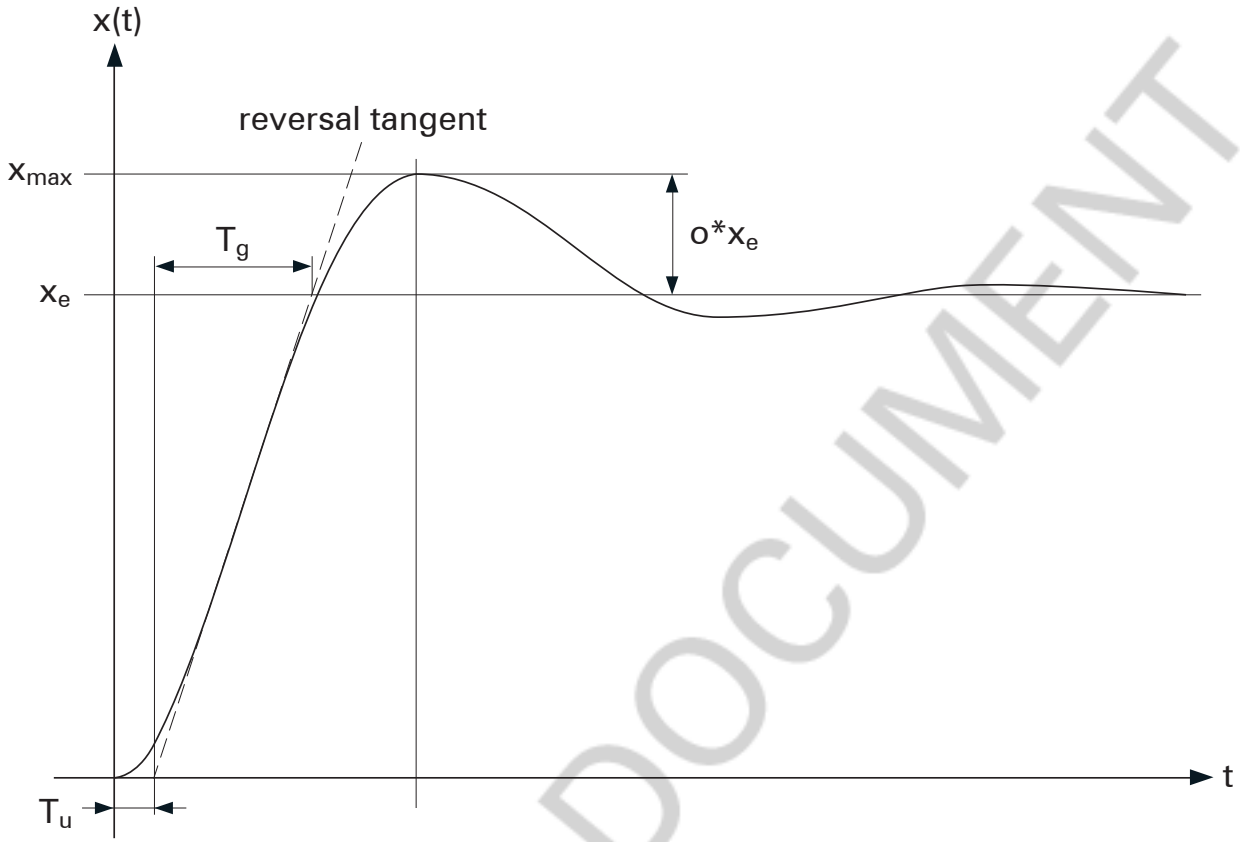


Fig. 31 Step response characteristics

- **Transient overshoot** o: difference between the maximum value of the step response and the final stationary value, based on the final stationary value (usually specified in percentage)
- **Dwell time** T_U: calculated from the intersection of the reversal tangent of the first rise with the time axis
- **Rise time** T_G: the time difference between the intersections of the reversal tangent of the first rise with the time axis and the final stationary value
- **Remaining control deviation** e_∞: the difference between the set value and final stationary value of the actual value

10.4.2 Characteristics in the frequency domain

The **gain crossover frequency** ω_C of the open loop is the intersection of the phase characteristic with the 0dB line. The gain crossover frequency divides the frequency domains that are amplified/weakened by the open loop.

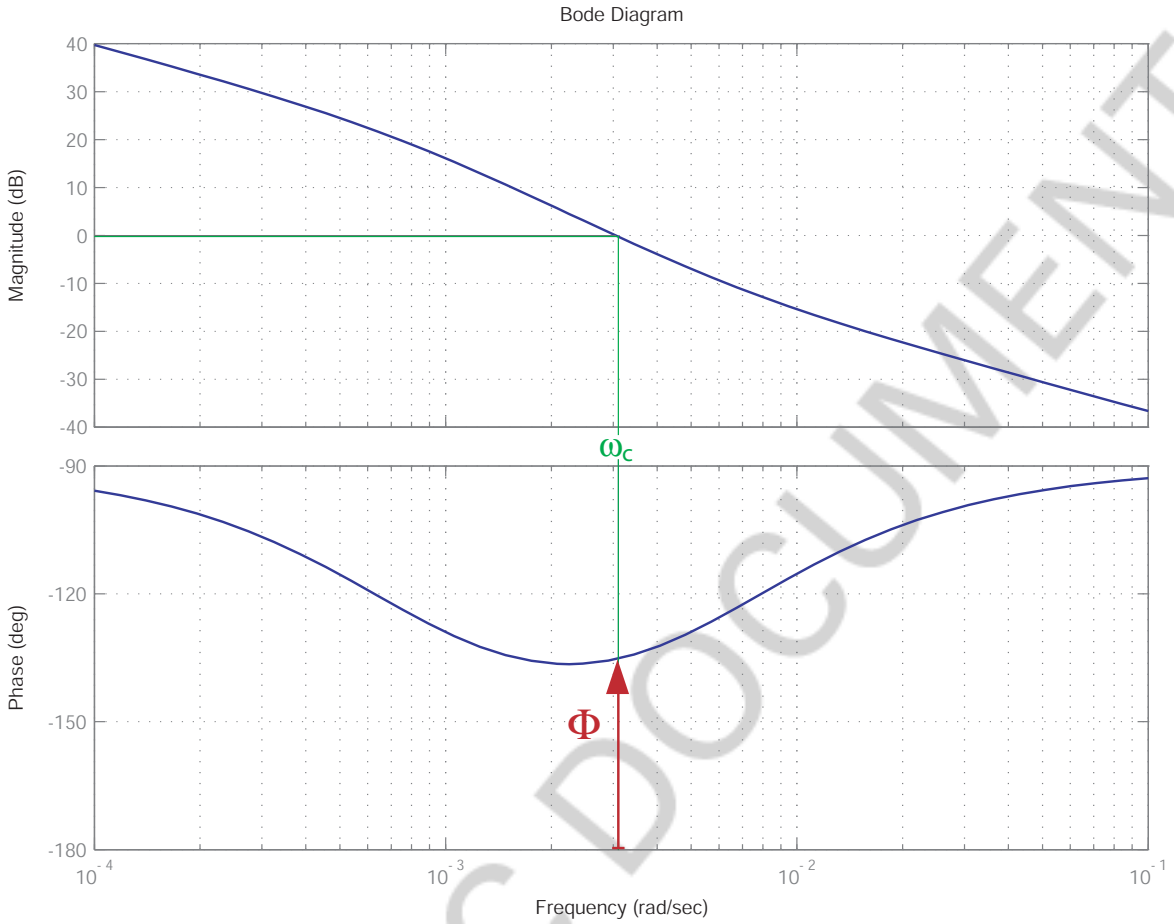


Fig. 32 Gain crossover frequency and phase margin

The **phase margin** Φ [°] of the open loop is of considerable importance to the stability of a control loop (with prevalent stable system to be controlled). The phase margin is the distance of the phase characteristic from -180° at the gain crossover frequency (Fig. 32):

$$\Phi = \arg(L(j\omega_C)) + 180° :$$

The closed control loop is stable if the phase margin Φ of the open loop is positive.

A thinking exercise: The linear open loop $L(j\omega)$ is excited using a sinusoidal input variable with angular frequency ω. The output variable then also begins to oscillate in a sinusoidal pattern with an identical angular frequency ω, but different amplitude and phase. The phase shift compared to the input variable is negative in real systems (with low pass character).

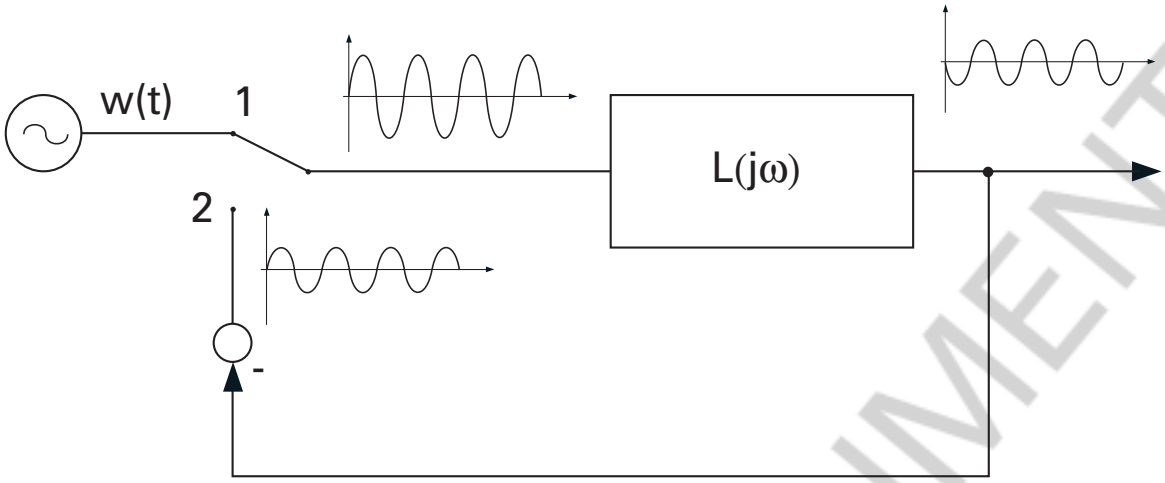


Fig. 33 For the thinking exercise

Now, if there is an angular frequency ω, for which the phase shift is exactly -180°, then the returned signal in item 2 has the same phase length at the summing point as the input signal in item 1 because of the signal inversion (due to the negative sign).

If the returned signal in item 2 also has the same amplitude as the input signal in 1, then the system does not detect any change when the switch is transferred, thereby causing the existing continuous oscillation to maintain itself. As a result, the feedback control system (the closed control loop) is located right at the stability limit.

If the gain of $L(j\omega)$ is less than 1 (i.e. $|L(j\omega_1)|<1$) at a phase shift of -180°, then the signal in item 2 is indeed in-phase with the input signal, but its amplitude is smaller. That is why the amplitude of the oscillation is decreased when the switch is transferred (closing the control loop). In this case, the closed control loop is stable.

If the gain of $L(j\omega)$ is greater than 1 (i.e. $|L(j\omega_1)|>1$) at a phase shift of -180°, then the signal in item 2 is indeed in-phase with the input signal, but its amplitude is greater. The amplitude of the oscillation is increased because of the feedback when the switch is transferred (closing the control loop). In this case, the closed control loop is instable.

10.4.3 Relationship between characteristics in the time and frequency domain

The following applies to the relationship between the phase margin of the open loop Φ and the transient overshoot ü of the closed control loop:

$$\Phi[°]+\ddot{u}[\%]\approx 70 \qquad \text{Eq. 6}$$

The phase margin of the open loop is a measurement for the oscillation tendency of the closed control loop and measures the distance to the stability limit.

The relationship shown above (Eq. 6) applies precisely to control loops, whose closed loop exhibits oscillation-capable PT_2 behavior. Experience has shown that this formula can also be applied as a guide for other control loops.

Fig. 32 shows gain crossover frequency and phase margin in the Bode diagram for the open loop from:

$$G(s) = \frac{0.085}{(1+1250 \cdot s) \cdot (1+500 \cdot s)},$$

and

$$R(s) = 88.61 \cdot \left(1 + \frac{1}{732 \cdot s} + 128 \cdot s\right).$$

The phase margin is $\Phi = 45°$. According to the empirical formula shown above, this would indicate an overshoot percentage of $ü \approx 25\%$. An actual overshoot of $ü = 29\%$ can be read from the step response in Fig. 34.

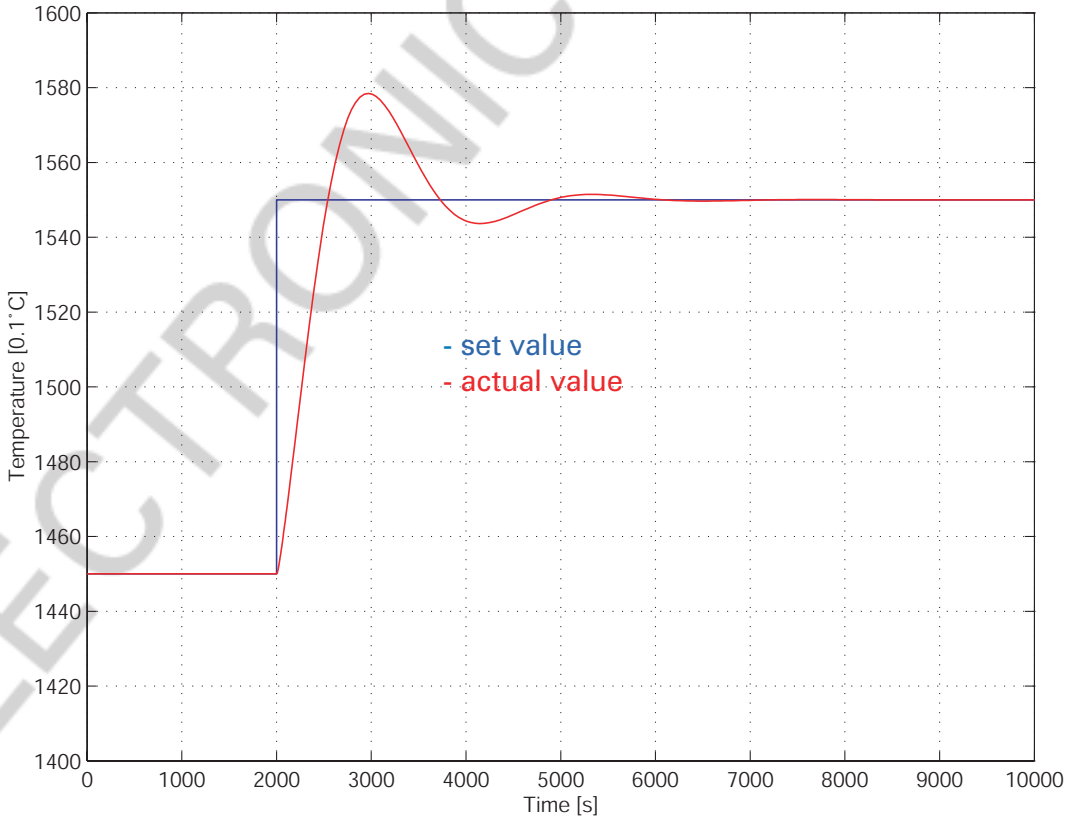


Fig. 34 Step response of the closed loop

The following applies to the relationship between the gain crossover frequency ω_C of the open control loop and the rise time of the closed control loop T_G:

$$\omega_C \cdot T_G \approx 1.5$$

The gain crossover frequency ω_C of the open control loop is an approximate measure for the spectrum of the open control loop and therefore also for the speed of the closed control loop.

The formula shown above has also proven itself in practical application, e.g. extract the gain crossover frequency ω_C = *0.00305* rad/s from Fig. 27 and the rise time T_G = 476 s from Fig. 29. The result is $\omega_C \cdot T_G = 1.451$ for this control loop, which is in strong agreement with the empirical formula shown above.

11. CONTROLLER AND CONTROLLER SETTING

11.1 PID controller

The controllers from the PID family (P/PI/PID) are the most important controller types for automating industrial processes. Nearly 95% of all industrial controlled systems can be sufficiently stabilized using a controller from this class.

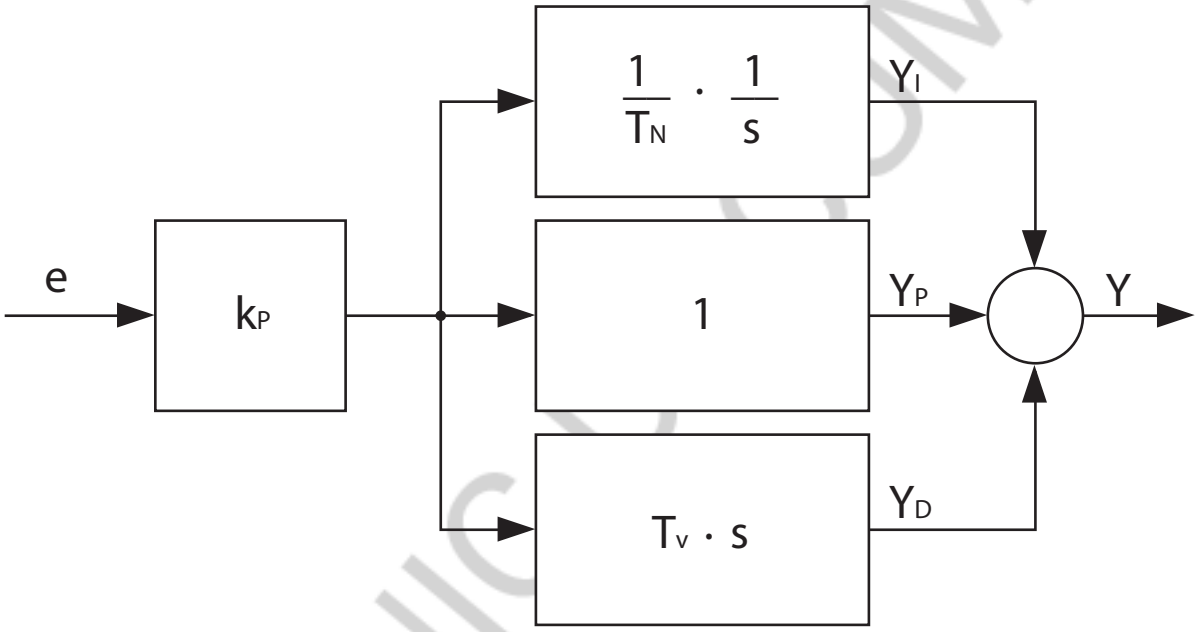


Fig. 35 Block diagram of the ideal PID controller

The following is an ideal transfer function for a PID controller:

$$R(s) = k_P + \frac{k_I}{s} + k_D \cdot s = k_P\left(1 + \frac{1}{T_N \cdot s} + s \cdot T_V\right).$$

The P-element produces a manipulated variable proportional to the control deviation *e(t)*:

$$Y_P(t) = k_P \cdot e(t).$$

The I-element produces a manipulated variable, which is proportional to the temporal integral of the control deviation:

$$Y_I(t) = \frac{k_P}{T_N}\int e(t) \cdot dt\,.$$

The integral action time T_N is the time span, which a constant control error must meet for the I-element to generate the same manipulated variable as the P-element.

If a system, which is to be controlled, does not contain an integral element, then a remaining control deviation can only be prevented using an I-element in the controller. An I-element lowers the stability of a control loop and causes overshoot. The smaller the integral action time, the stronger the effect of the I-element.

The D-element produces a manipulated variable, which is proportional to the temporal derivative of the control deviation:

$$Y_D(t) = k_P \cdot T_V \cdot \dot{e}(t).$$

The derivative action time T_V indicates the time span, which an increasing control deviation of 0 with a constant gradient must meet for the P-element to generate the same manipulated variable as the D-element.

A D-element increases the speed and improves the stability of a control loop. A larger integral action time increases the effect of the D-element.

The proportional gain k_P influences all three elements of a PID controller and is decisive in determining the dynamics and oscillation-tendency of a control loop. The rise time becomes smaller if the proportional gain is increased (faster). The phase margin becomes smaller (destabilizing) if the gain crossover frequency is increased (faster).

The D-element is generally implemented as filtered derivative unit because an ideal derivative unit cannot be realized. The following is the transfer function of a true PID controller:

$$R(s) = k_P\left(1 + \frac{1}{T_N \cdot s} + \frac{s \cdot T_V}{1 + s \cdot T_F}\right).$$

Fig. 37 shows the Bode diagram of an ideal and true PID controller.

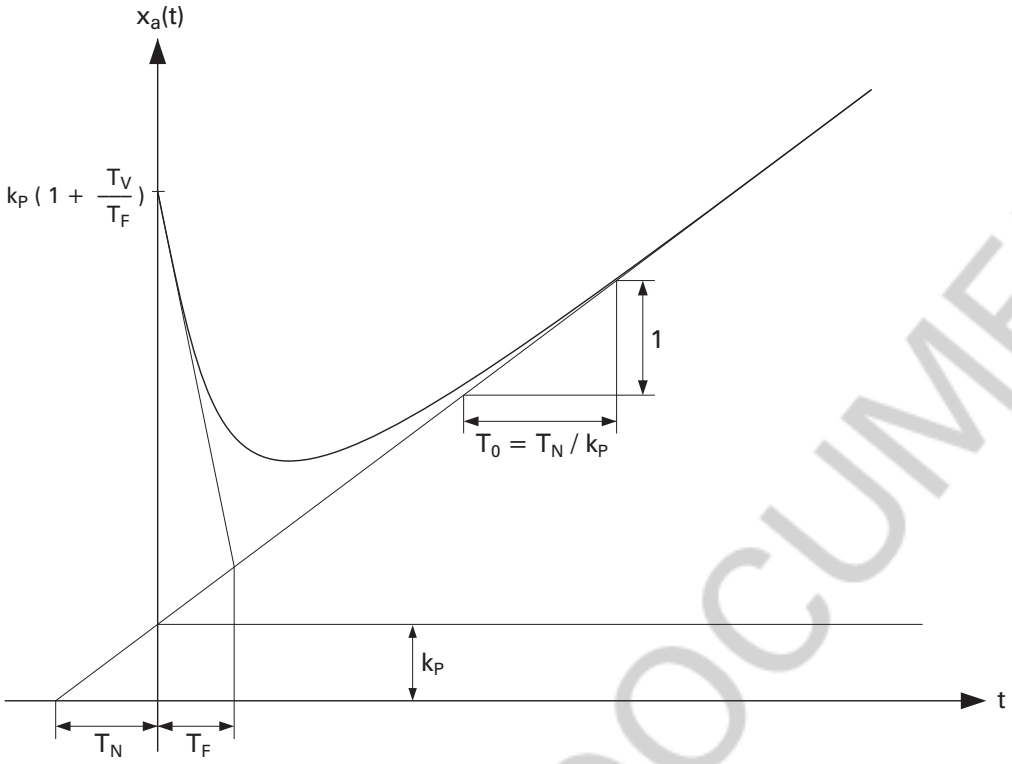


Fig. 36 Step response of a true PID controller

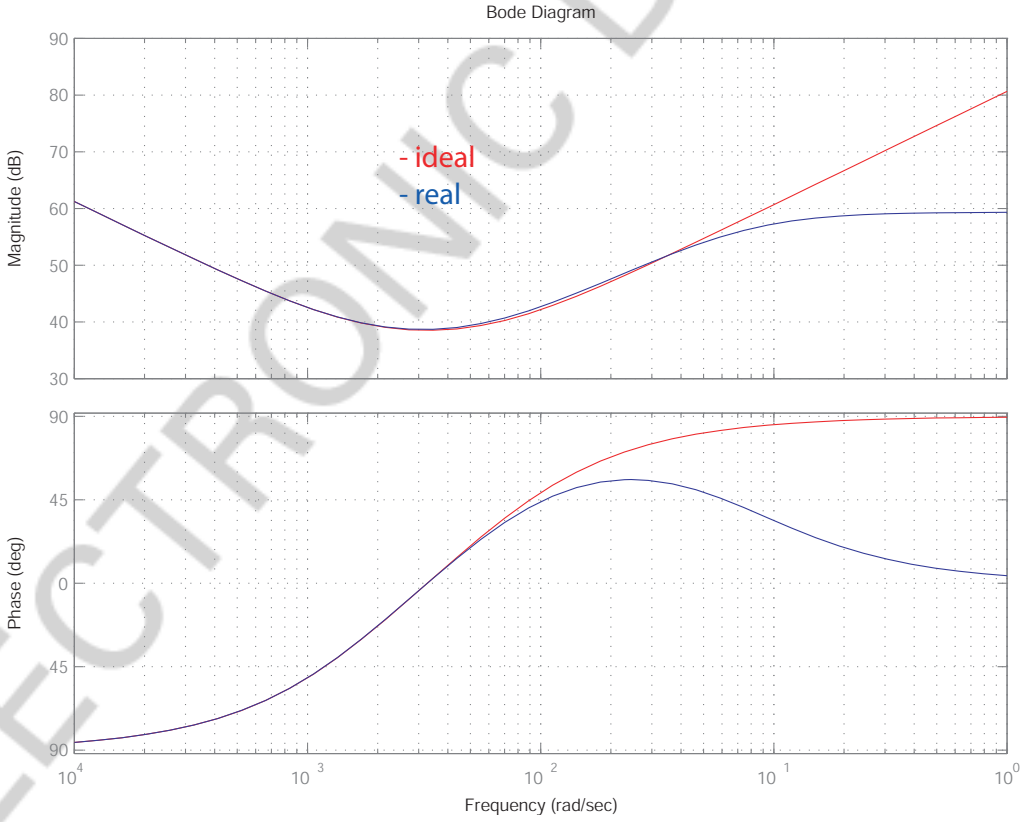


Fig. 37 Bode diagram of a true PID controller

The PID controller function blocks LCPID() and LCRPID() in the Automation Studio control technology libraries contain numerous extensions that are important for practical application (anti-windup, consideration of feed-

forward variables, differentiator mode, etc). Detailed information can be found in the extensive documentation in Automation Studio's online help files.

11.2 Controller setting

11.2.1 Setting guidelines according to Chien, Hrones and Reswick

The setting guidelines lines according to Chien, Hrones and Reswick are suitable for controlled systems that are not capable of oscillation with first or higher-order delay and an additional dead time. A first-order delay element with dead time is used as system model:

$$G(s) = \frac{k_S}{1 + T_G s} \cdot e^{-sT_U}$$

The three parameters system gain k_S, rise time T_G and dead time (dwell time) T_U are determined (as shown in section 9) from the step response of the open loop ('process reaction curve') according to the reversal tangent procedure.

The control parameters are determined depending on the system parameters, the control type being used and the desired overshoot behavior ($o \sim 0\%$ is equal to a non-periodic behavior) with the following:

$$k = \frac{T_G}{k_S \cdot T_U} = \frac{\Delta Y}{\dot{T}_{\max} \cdot T_U} :$$

Controller		Set-point tracking design		Disturbance rejection design	
		$o \sim 0\%$	$o \sim 20\%$	$o \sim 0\%$	$o \sim 20\%$
P	k_P	$0.3 \cdot k$	$0.7 \cdot k$	$0.3 \cdot k$	$0.7 \cdot k$
PI	k_P	$0.35 \cdot k$	$0.6 \cdot k$	$0.6 \cdot k$	$0.7 \cdot k$
	T_N	$1.2 \cdot T_G$	T_G	$4 \cdot T_U$	$2.3 \cdot T_U$
PID	k_P	$0.6 \cdot k$	$0.95 \cdot k$	$0.95 \cdot k$	$1.2 \cdot k$
	T_N	T_G	$1.35 \cdot T_G$	$2.4 \cdot T_U$	$2 \cdot T_U$
	T_V	$0.5 \cdot T_U$	$0.47 \cdot T_U$	$0.42 \cdot T_U$	$0.42 \cdot T_U$

In the above table it is evident that all three parameters of the PT_1T_T system model are necessary for a set-point tracking design.

In accordance to the setting guidelines of Chien, Hrones and Reswick, the controllers R1 (non-periodic disturbance rejection design) and R2 (non-periodic set point tracking design) from section 10.3.3 were calculated for the system approximation:

$$\hat{G}(s) = \frac{0.085}{1 + 2300 \cdot s} \cdot e^{-305 \cdot s}$$

of the system to be controlled:

$$G(s) = \frac{0.085}{(1 + 1250 \cdot s) \cdot (1 + 500 \cdot s)} \cdot e^{-90 s}$$

from section 9.3. The respective responses from the closed control loops to jump-causing changes in the reference and disturbance variables are shown in Fig. 29 and Fig. 30.

11.2.2 Setting guidelines according to Ziegler and Nichols

Ziegler and Nichols are the pioneers of control setting procedures and published a method in 1942 for setting PID controllers in the closed loop based on empirical analyses:

- The controller in the control loop is initially operated as true P controller, whereby the controller gain k_P is increased up to the value k_{crit}, at which point the control loop reaches the stability limits and sets a stationary continuous oscillation.
- The period duration T_{crit} of the continuous oscillation is measured.
- The controller settings are produced according to the controller type:

Controller	k_P	T_N	T_V
P	$0.5 \cdot k_{crit}$	-	-
PI	$0.45 \cdot k_{crit}$	$0.85 T_{crit}$	-
PID	$0.6 \cdot k_{crit}$	$0.5 T_{crit}$	$0.125 T_{crit}$

This method has a decay rate of the transient overshoot of $D = 25\%$ to the target. That means that the transient overshoot of a period i to the next period $i+1$ decays according to:

$$\frac{o_{i+1}}{o_i} = D = 0.25$$

This method is suited for designing disturbance variable controllers, which generally exhibit high overshoot when jumps occur in the reference variable.

11.2.3 Design in the Bode diagram

When designing the controller in the Bode diagram (frequency characteristic method), the relationships between characteristics in the time and frequency domain (section 10.4.3) are used to design a controller for a desired set point tracking behavior (rise time, transient overshoot).

In comparison to the setting guidelines, this method fulfills specifications with a high-degree of accuracy. Extensive literature is available providing further details about this method.

11.3 Autotuning procedure

An autotuning procedure is a combination of inter-coordinated identification and controller setting procedures, which run automatically and are controlled by algorithms. They are the most convenient method of controller setting for the user.

A stimulating input signal is first actuated on the system and the system's response is recorded. The system's transfer function is determined from the comparison of these input and output signals. A controller is then calculated for this system in such a way so that the closed control loop exhibits the desired behavior. After setting the parameters once, these procedures will run online fully automatically without intervention from the user and can be repeated at any time.

The function blocks LCPIDTune and LCRPIDTune in Automation Studio's control-technology libraries LoopCont and LoopConR provide two different autotuning procedures:

- Oscillation attempt: uses a periodic square-wave signal as system excitation in the closed control loop for setting P/PI/PID controllers and features the easiest method for setting parameters.
- Step response: uses a manipulated variable jump as system excitation in the open control loop and offers a multitude of possibilities for controller settings according to the desired controller type (P/PI/PID) and behavior (disturbance rejection or set-point tracking design, transient overshoot).

Detailed information can be found in the extensive documentation in Automation Studio's online help files.

12. SUPPLEMENTS

12.1 The influence of dead time

The transfer function G(s) is expanded by one dead time element with the dead time T_t to analyze the influence of dead times:

$$G_t(s) = G(s) \cdot e^{-j\omega T_t}$$

Dead times in systems do not change the magnitude in the frequency characteristic, because the following applies:

$$|G_t(j\omega)| = |G(j\omega)|,$$

however an additional phase rotation around $-\omega T_t$:

$$\arg(G_t(j\omega)) = \arg(G(j\omega)) - \omega T_t$$

Because the phase is decayed linearly with ω due to the dead time, which causes a reduction of the open loop's phase margin, every control loop subject to dead time becomes instable at a specific amount of gain – this becomes more frequent the larger the dead time is.

Fig. 38 shows the Bode diagram of the system transfer function:

$$G(s) = \frac{0.085}{(1+1250 \cdot s) \cdot (1+500 \cdot s)} \cdot e^{-T_U s}$$

for four different values of $T_U = 0, 90, 180$ and 270 s. Fig. 39 shows the respective Bode diagram of the open control loop for the above systems with the controller:

$$R(s) = 88.61 \cdot \left(1 + \frac{1}{732 \cdot s} + 128 \cdot s\right).$$

The phase margin of the open loop is $\Phi = 45, 29, 13$ and -3°.

The step responses of the corresponding closed control loops are shown in Fig. 40. It is clearly evident that the controller gain for the highest value of the system dead time $T_U = 270$ s is already beyond the stability limits.

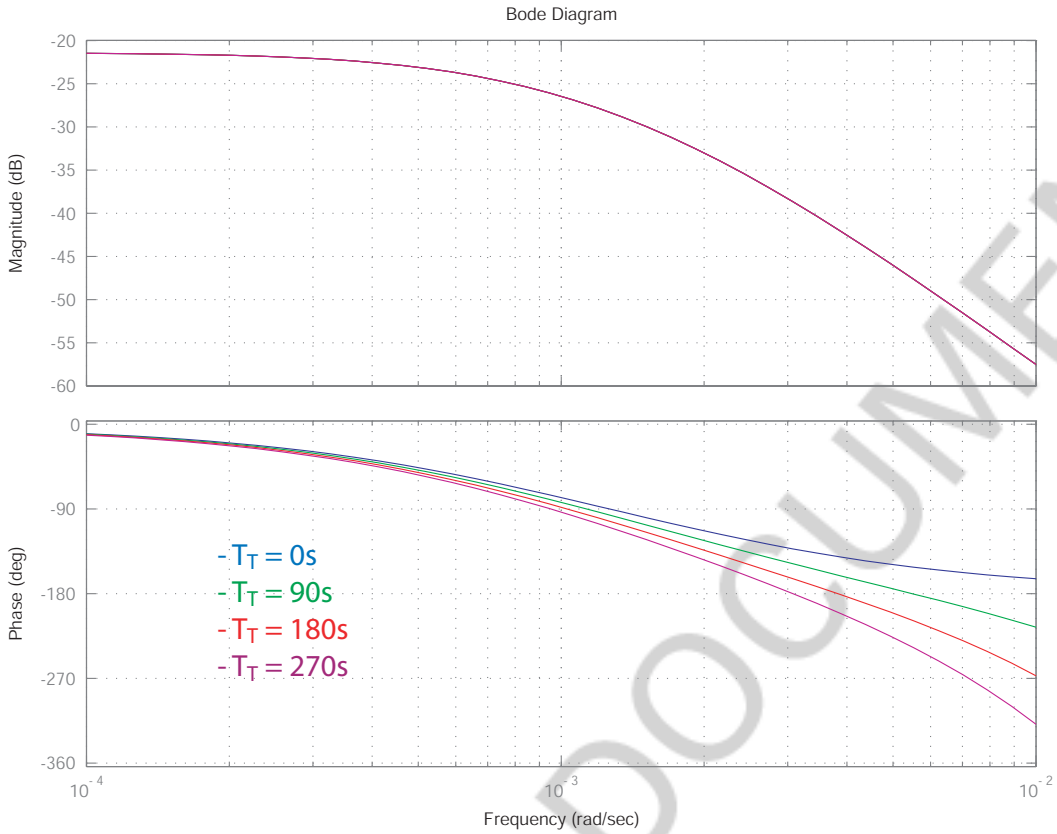


Fig. 38 Bode diagrams of a system transfer function with different dead times

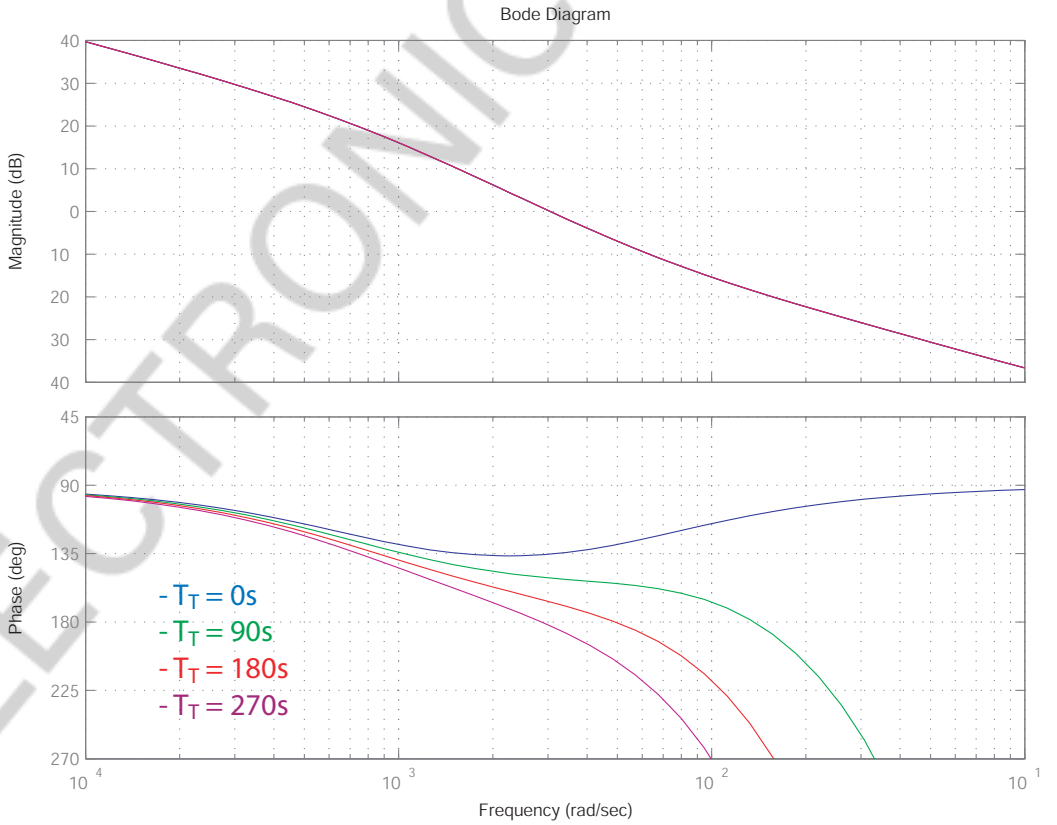


Fig. 39 Bode diagrams of an open control loop with different dead times

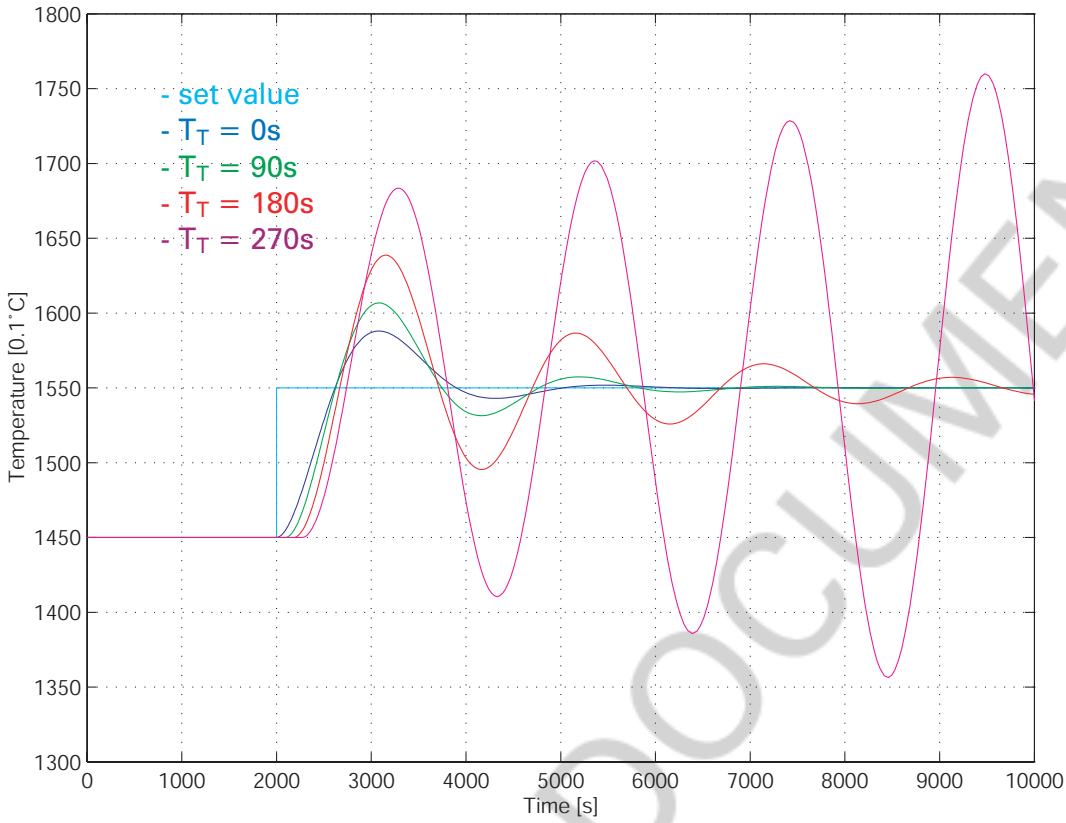


Fig. 40 Step responses of the closed control loop with different dead times

12.2 The influence of measurement errors

Fig. 41 shows a control loop with measurement error. The transfer function of the measurement error signal is:

$$T_n(s) = \frac{x(s)}{n(s)} = \frac{-R(s)G(s)}{1+R(s)G(s)} = -T(s)$$

and therefore corresponds exactly to the negative reference transfer function. Thus, the controlled variable is calculated as follows:

$$x(s) = T(s) \cdot w(s) + T_n(s) \cdot n(s) = T(s) \cdot [w(s) - n(s)].$$

Measurement errors cannot be compensated for using a controller. Measurement errors act like a changed reference variable and result in a remaining control deviation.

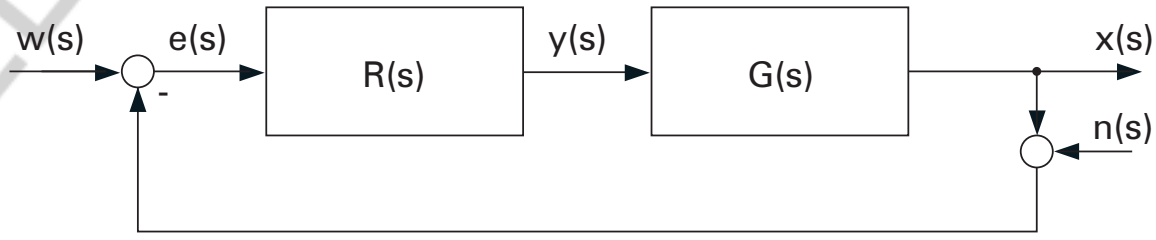


Fig. 41 Control loop with measurement error

12.3 Mixed control loop

12.3.1 Disturbance variable feedforward

If disturbance variables occur in the standard control loop, then the controller generates a corrected manipulated variable if the controlled variable has already been changed.

If a disturbance variable can be measured and the influence on the control loop (the transfer function $D(s)$) is known, then the disturbance can be compensated for by implementing a feedforward of an additional manipulated variable y_d. This generally will not eliminate the influence of disturbances completely, but will often reduce it considerably.

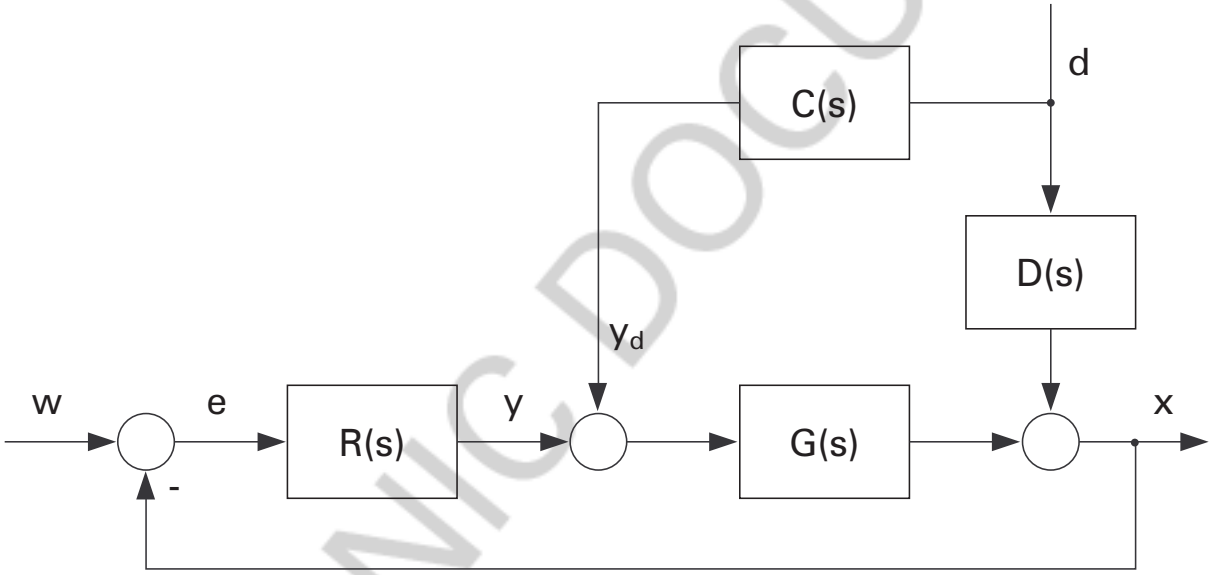


Fig. 42 Control loop with disturbance variable feedforward

A perfect disturbance variable compensation has the following form:

$$C(s) = -\frac{D(s)}{G(s)}.$$

12.3.2 Set value feed-forward

If the respective manipulated variables are known for a system to be controlled over a wide range of values of the controlled variable then this manipulated variable (as function of the controlled variable) can additionally be fed forward to the system in order to maintain the system in a stationary state at these values of the controlled variable (e.g. from an analysis like in section 8.2 and 8.3 or from empirical observation of the system to be controlled). This set value feed-forward improves the control loop's set-point tracking behavior.

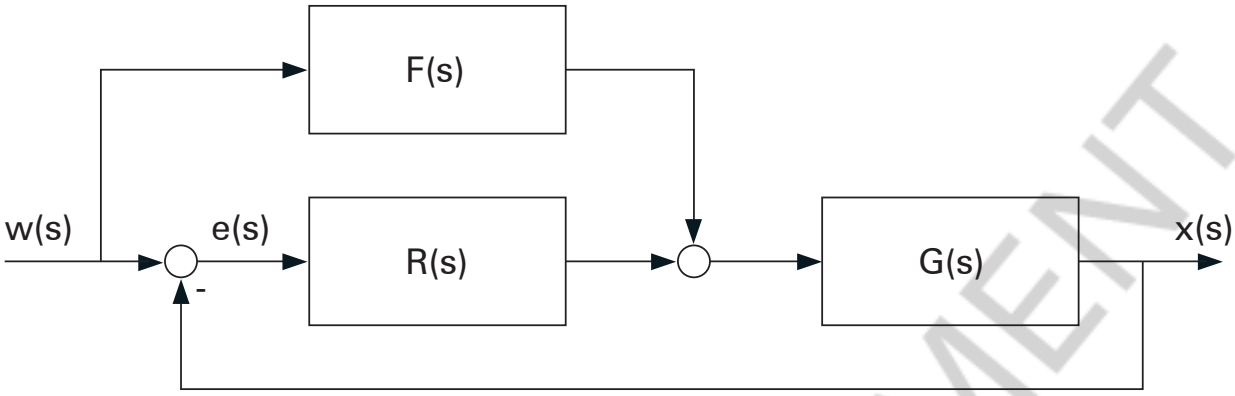


Fig. 43 Control loop with set value feed-forward

12.3.3 Cascade control

Cascading the control loop makes sense if a system that must be controlled can be divided into multiple subsystems connected in series, which have different system dynamics (different speeds) and whose output variables can be measured.

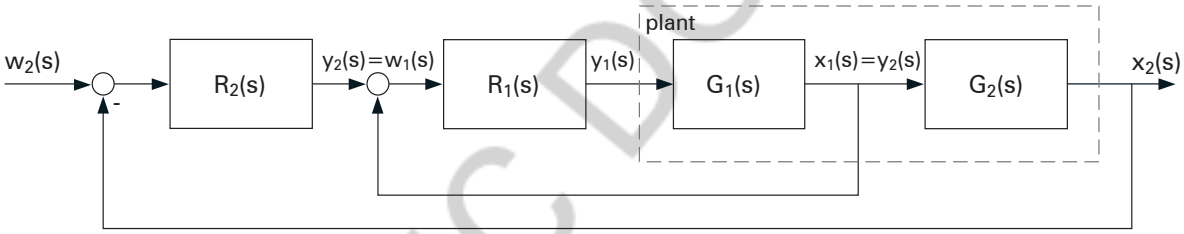


Fig. 44 Cascade control

The reference transfer function of the inner loop is:

$$T_1(s) = \frac{R_1(s) \cdot G_1(s)}{1 + R_1(s) \cdot G_1(s)}.$$

The reference transfer function of the outer loop (the entire cascade) is:

$$T_2(s) = \frac{R_2(s) \cdot T_1(s) \cdot G_2(s)}{1 + R_2(s) \cdot T_1(s) \cdot G_2(s)}$$

The outer control loop specifies the set value for the inner control loop, which has faster system dynamics. The controller design and practical startup are both performed from the inside to the outside.

Advantages of cascading control loops:

- Improved dynamic behavior (the fast inner control loop is completely unaffected by the slow dynamics of the outer loop)

- Limits are easily implemented because the set values correspond to the controller outputs of the higher-level controller
- Disturbances in the fast inner loop hardly influence the slower outer loop

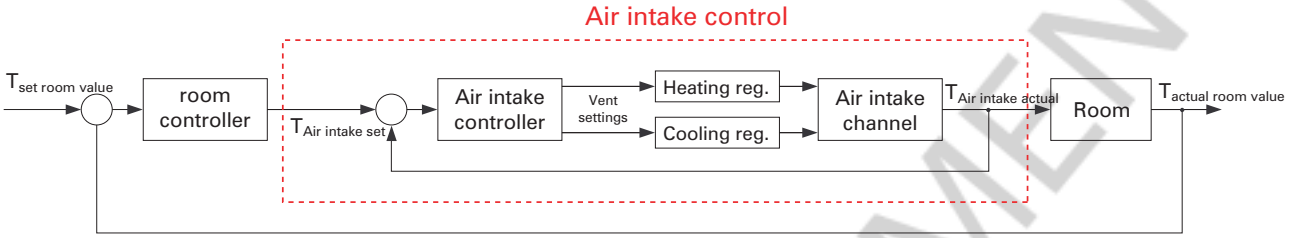


Fig. 45 Cascaded room temperature control

Closed-loop controls for drive systems generally have a triple cascade structure. The control concept of the ACOPOS™ servo drive is a topic of the training module TM450.

12.4 Prefilter

A control loop cannot meet the highest demands in regard to set-point tracking behavior and disturbance rejection behavior with the structure of the standard control loop as shown in Fig. 27 and in section 10.3.

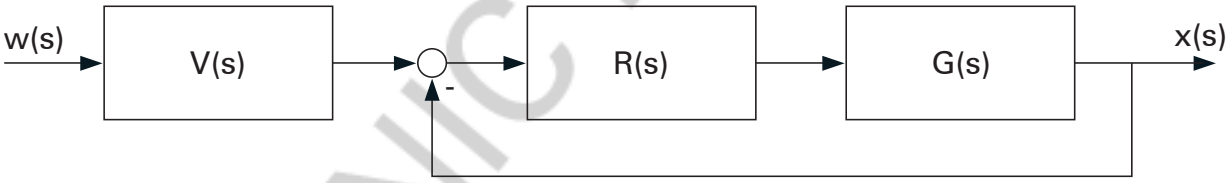


Fig. 46 Control loop with prefilter

Fig. 46 shows a control loop with prefilter. The job of the prefilter V(s) is to prevent high overshoot when reference variable jumps occur by using a low-pass filter of the reference signal. The reference transfer function is then calculated as follows:

$$T(s) = \frac{V(s) \cdot L(s)}{1 + L(s)} = \frac{V(s) \cdot R(s) \cdot G(s)}{1 + R(s) \cdot G(s)}.$$

The disturbance variable transfer function remains unchanged compared to the standard control loop:

$$T_d(s) = \frac{1}{1 + R(s) \cdot G(s)}.$$

In this case, the controller R(s) is strongly set for disturbance variable suppression (comparatively low phase margin of the open loop).

Fig. 47 shows the step responses of a control loop with a first-order low-pass as prefilter:

$$V(s) = \frac{1}{1 + T_F \cdot s}$$

for various filter time constants $T_F = 0, 650$ und 850 s.

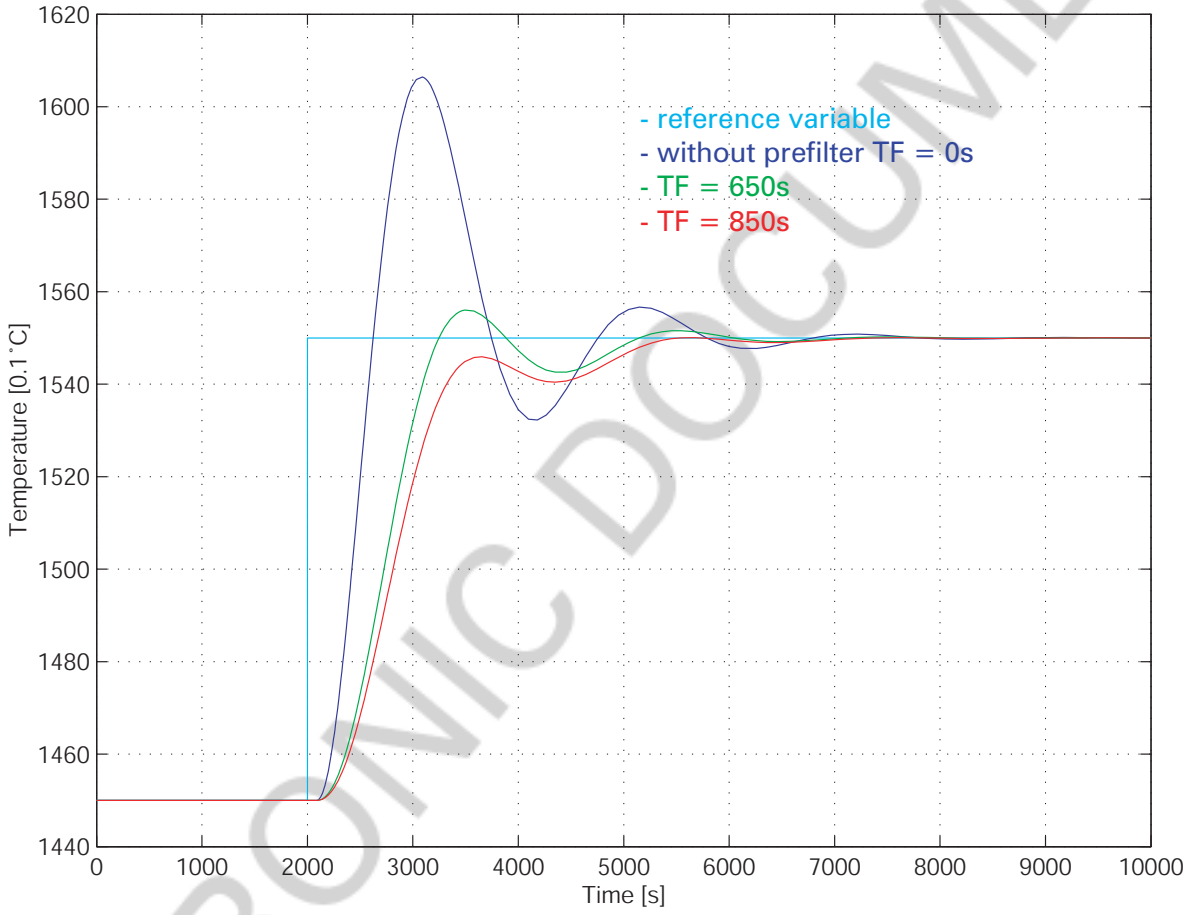


Fig. 47 Step response of control loop with prefilter

12.5 Non-linearities

Non-linear systems have different transfer behavior at every operating point. The non-linearity can appear in the control action or in the autonomous system dynamics (or in both of course).

12.5.1 Non-linearity in the control action

One example of non-linearity in the control action is a feed-through cooling system, in which the valve lift (opening) is used as control action.

If the non-linear characteristic of the control action is known well-enough, then it can be compensated for by connecting the inverse characteristics in the controller (Fig. 48).

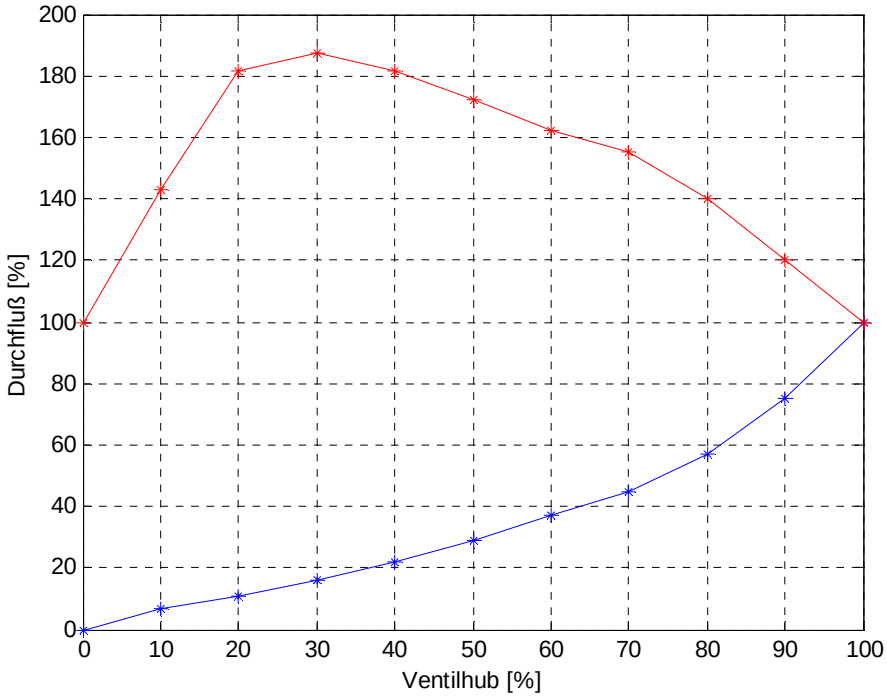


Fig. 48 Non-linear characteristic and its inversion

A non-linearity, which occurs in every true control system and which cannot be compensated for is the manipulated variable limitation because real actuators only allow limited control actions.

12.5.2 Linearization around a operating point

In the area of a specific operating point (provided by a specific value of the controller and manipulated variables), every non-linear system behaves in a linear manner for minor deviations from this working point.

12.5.3 Non-linearity in the autonomous system dynamics

Heat loss via radiation in the extruder zone from section 2 is an example of non-linearity in the autonomous system dynamics. If the non-linear radiation level is taken into consideration, then the following calculation is produced for the transfer function in the environment of the temperature T_S for an ambient temperature of $T_0 = 25°C$:

$$G(s) = \frac{k_S}{1 + T_G s}$$

with the system gain [°C/W]:

$$k_S = \frac{1}{\alpha \cdot A + 4 \cdot \varepsilon \cdot \sigma \cdot A \cdot T_S^3}$$

and the time constant [s]:

$$T_G = \frac{m \cdot c}{\alpha \cdot A + 4 \cdot \varepsilon \cdot \sigma \cdot A \cdot T_S^3} = m \cdot c \cdot k_S .$$

The effect of this non-linearity is that an increasing temperature T_S causes the system gain to decrease and the system time constant to become smaller. The heating control action is needed to keep the extruder zone at the stationary temperature T_S :

$$\dot{Q}_H(T_S) = \alpha A(T_S - T_0) + \varepsilon\sigma A(T_S^4 - T_0^4)$$

Fig. 49 shows the characteristic curve of $\dot{Q}_H(T_S)$ for an industrial extruder zone.

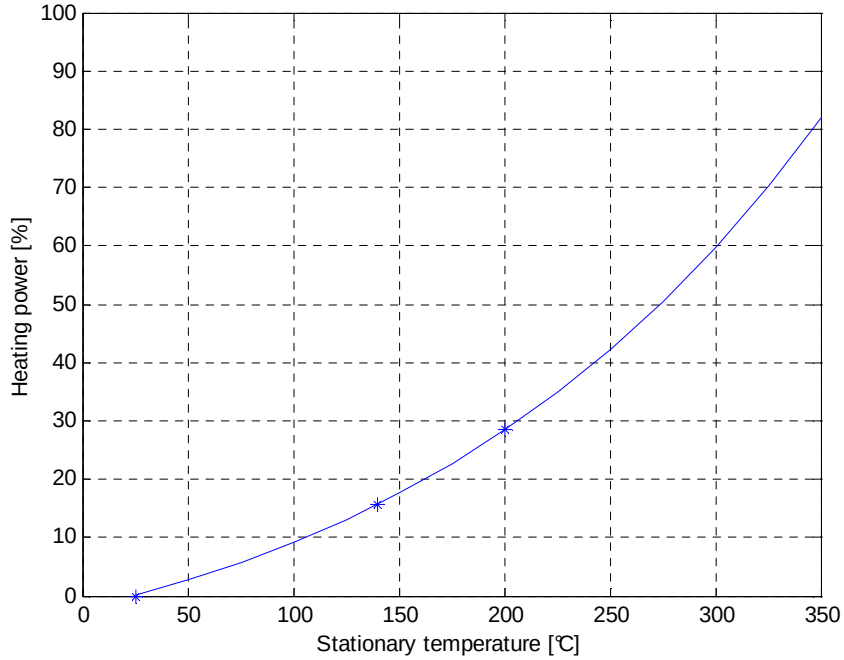


Fig. 49 Heating control action as function of the stationary temperature

12.6 Pulse width modulated actuator signals

Controlled systems with sufficient low pass character (e.g. thermal systems with large time constants) usually connect the output variable of the controller to the controlled system as digital signal via a pulse width modulation (PWM) instead of an analog connection.

This makes it possible to use fast-switching digital actuators (e.g. solid state relay), which are much more economical than analog actuators (e.g. heating elements).

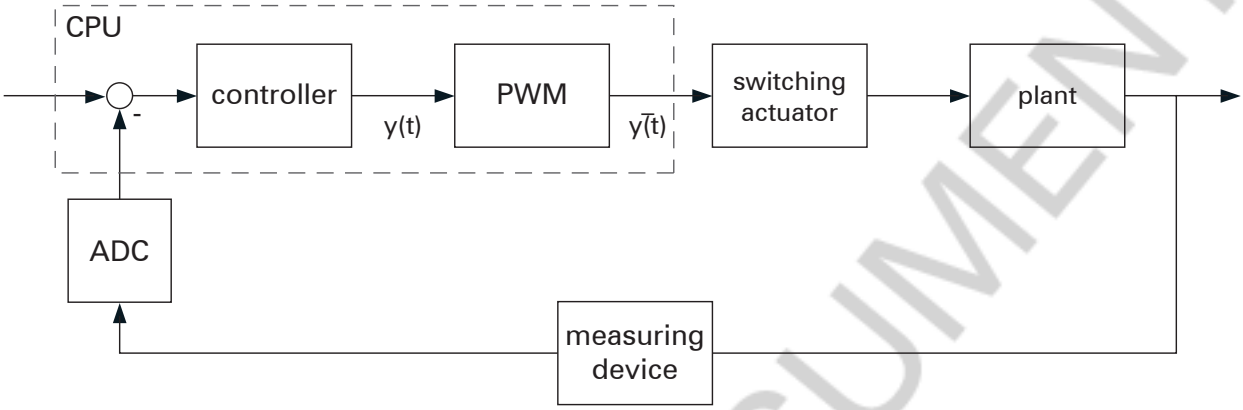


Fig. 50 Control loop with pulse width modulated manipulated variable

To prevent the switching processes of the manipulated variable signal from affecting the control variable, they must occur at such a high rate of frequency that they are sufficiently damped by the system to be controlled,

$$\frac{|G(j\omega_S)|}{k_S} << 1,$$

whereby k_S the stationary gain of the system to be controlled,

$$\omega_S = 2 \cdot \pi \cdot T_S$$

the switching frequency and T_S are the period duration of the PWM signal. That means that the period duration of the PWM signal must be selected much smaller (recommended value: factor 0.1) than the fastest system time constant.

When dealing with actuator elements that are operated on an AC network, make sure that inaccuracies do not occur due to the switching characteristic of the actuator elements in relation to the cycle time of the task for the modulated control of the actuator elements.

Semiconductor relays that are designed as zero-voltage switches are an important example of this. On an AC voltage source, these relays switch on only when the voltage crosses zero and off only when the current crosses zero.

As a result of this switching characteristic, there can be considerable differences on an AC network between manipulated variables calculated in the software and the physical manipulated variable actually connected to the system to be controlled, whereby the accuracy of the control loop is reduced.

The inaccuracies have an even greater effect, the more exactly the period durations of the voltage supply match the period durations of the modulation (and therefore the cycle time of the task class in which the modulation procedure is processed).

That is why task class cycle times starting at approx. 100 ms should be used for the modulated control of heating relays on AC networks with a mains frequency of 50 - 60 Hz (equal to a period duration of 16 – 20 ms).

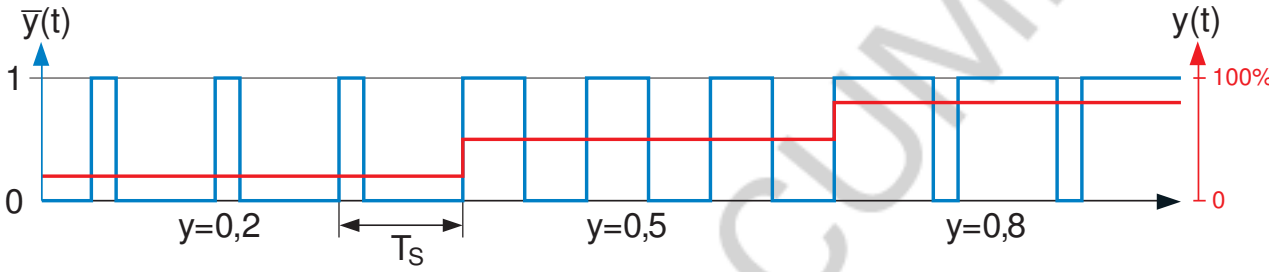


Fig. 51 PWM signal

12.7 Sampling control loops

Up to now, all of our observations have required that all signals in the control loop are continuous time functions, which can accept any real numerical values.

In reality, the CPU processes the control algorithm only at specific discrete points in time depending on the cycle time of the task class (sampling time), in which the control program is located. The measurement signals from the input modules and the manipulated variables are also sampled / written to the output modules at discrete cycle points in time.

As a result, information is 'given away' (temporally located between the sampling time points). Therefore, an event in the system sometimes cannot be reacted to until the next sampling instant. This reduces the quality of the dynamics and destabilizes the control loop (reduces the phase margin).

A far-reaching and highly effective theory (of discrete-time systems) exists for sampling control loops regarding analysis and design.

If sampling controllers are designed using the methods presented in this document for continuous dynamic systems, then the system to be controlled must be sampled fast enough to be able to neglect the destabilizing effects.

As a rule of thumb, it can be assumed that the cycle time T_A of the control task has to meet the demand:

$$T_A < \frac{T_{min}}{10},$$

whereby T_{min} is the smallest system time constant of interest.

Example: A mechanical construction has three resonant frequencies ($f_1 = 2.0$ Hz, $f_2 = 4.2$ Hz and $f_3 = 8.7$ Hz). f_1 and f_2 should be cancelled out using a controlled active damper. The cycle time of the control task is calculated as follows:

$$T_A < \frac{1}{10 \cdot f_2} = 0.0238 \text{ s}$$

Furthermore, every digital computer has just one finite computational accuracy. Additionally, quantizations (truncations) are created by the D/A and A/D conversion of the measurement and manipulated variables. The magnitude of the quantization error is determined by the resolution of the converter in the I/O modules.

Fig. 52 shows a control loop with a digital computer.

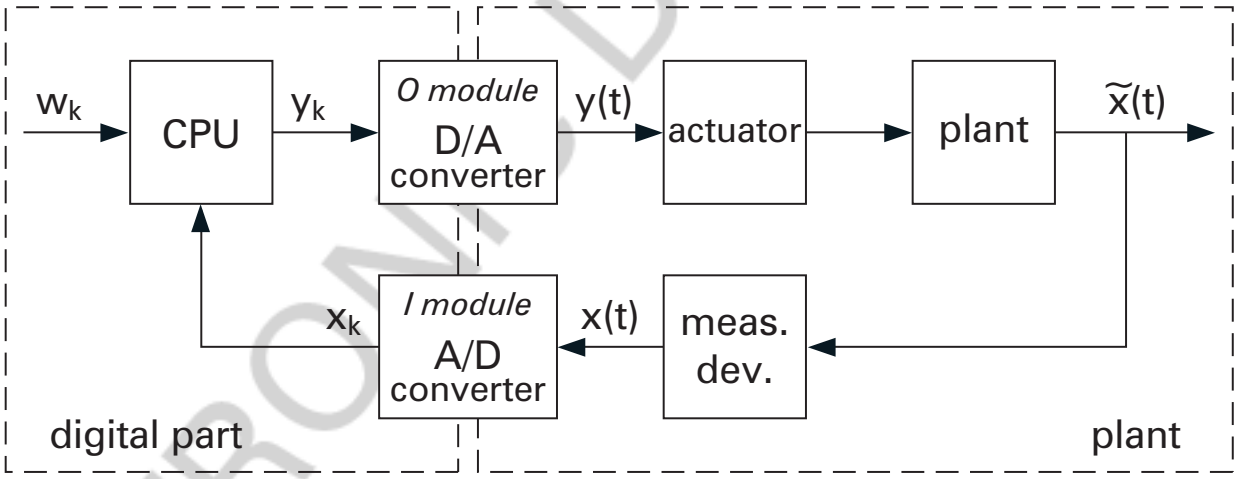


Fig. 52 Sampling control loop

13. PROCEDURE FOR SOLVING CONTROL TASKS

- What is/are the controlled variable(s)?
- What is/are the manipulated variable(s)?
- Construct a block diagram.
- How might the physical relationship between the manipulated variables and the controlled variables look? What kind of transfer function might the system have?
- Can the system display instable behavior?
- Record step-responses for the open loop.
- Analyze the step-responses.
- Calculate the controller settings.
- Test the controller settings by recording step-responses at various operating points in the closed loop.

14. SUMMARY

This training module discussed how to solve control tasks with the help of the Automation Studio library LOOPCONR.

After a brief introduction to the subject area, the practical application-oriented Part I worked to solve a few practical examples in the area of temperature control using the controller and autotuning function blocks.

In Part II, the topic of closed loop control was approached systematically to gain a theoretical knowledge base for better understanding and overcoming the demands that appear in practical application.

15. APPENDIX

15.1 LOOPCONR function block overview

Controller function blocks and tuning procedure:

Call	Description
LCRDblActPID()	PID controller with two outputs to control opposing actuator elements and perform integrated tuning
LCRPID()	PID controller
LCRPIDpara()	Manual configuration of the PID controller
LCRPIDTune()	Automatically determines the control parameters with various methods and setting guidelines
LCRSlimPID()	PID controller with integrated tuning

Controller function blocks and tuning processes especially for temperature systems:

Call	Description
LCRTempTune()	Optimized tuning procedure especially for temperature systems
LCRTempPID()	PID controller especially for temperature systems

Modulators:

Call	Description
LCRPFM()	Pulse frequency modulator
LCRPWM()	Pulse width modulator

Signal processing

Call	Description
LCRDifferentiate()	Derivative unit with filter
LCRIntegrate()	Integrator with limits and set value
LCRLimit()	Limiter with overrun indicators
LCRLimScal()	Scaling and limiting of REAL signals
LCRMinMax()	Smallest and largest peak value
LCRMovAvFlt()	Floating average value filter
Call	**Description**
LCRPT1e()	First-order delay element
LCRPT2()	Second-order delay element
LCRTt()	Dead time element
LCRScal()	Scaling of REAL signals

Function block for creating characteristic curves

Call	Description
LCRCurveByPoints()	y = f(x) function using coordinates

Other function blocks

Call	Description
LCRContinServo()	Control for a continuous servo drive
LCRRamp()	Ramp generator
LCRSimModExt()	Simulation model of an extruder with heating zones and cooling circulation

15.2 Solutions to the tasks

15.2.1 Task: LCRSlimPID() P-controller

Ladder diagram: LCRSimModExt()

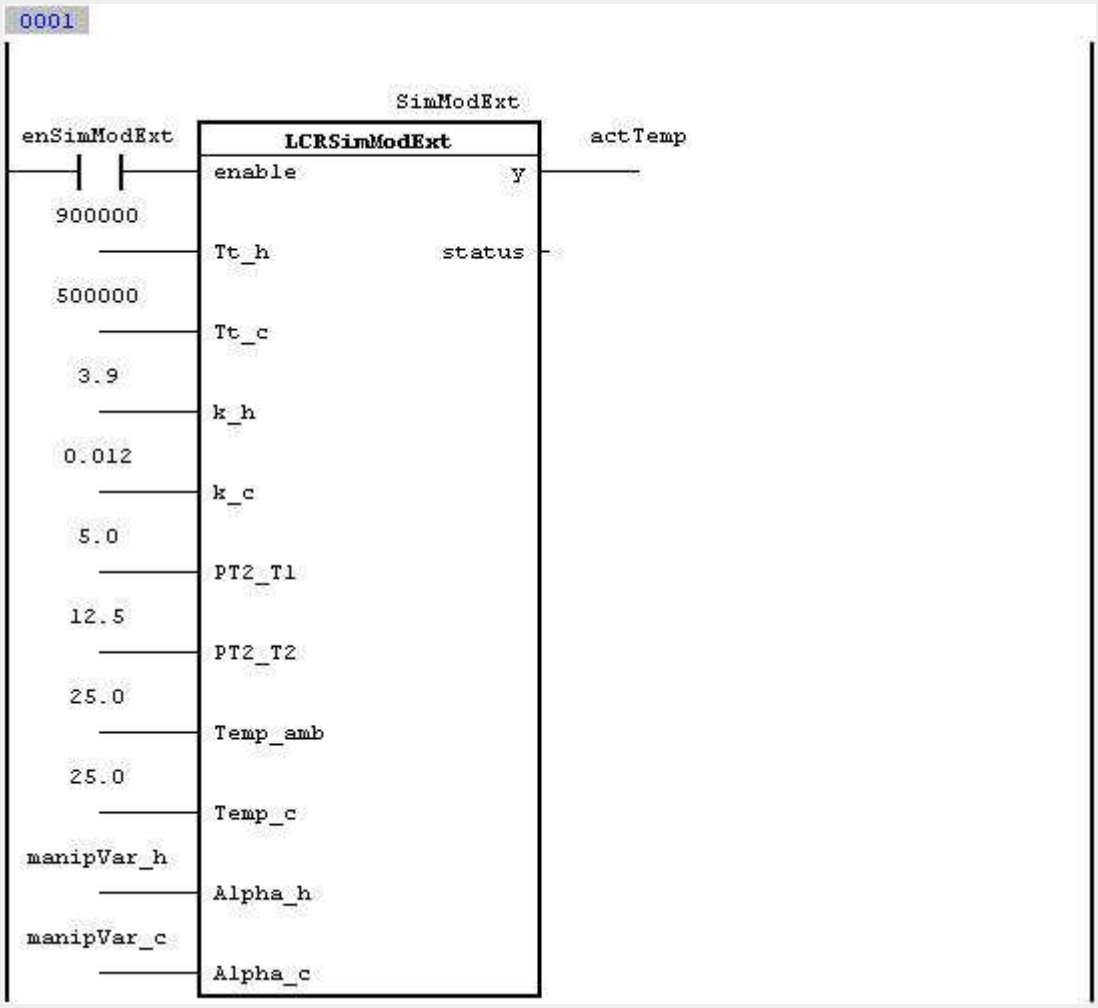


Fig. 53: LCRSimModExt() function block

Ladder diagram: LCRSlimPID()

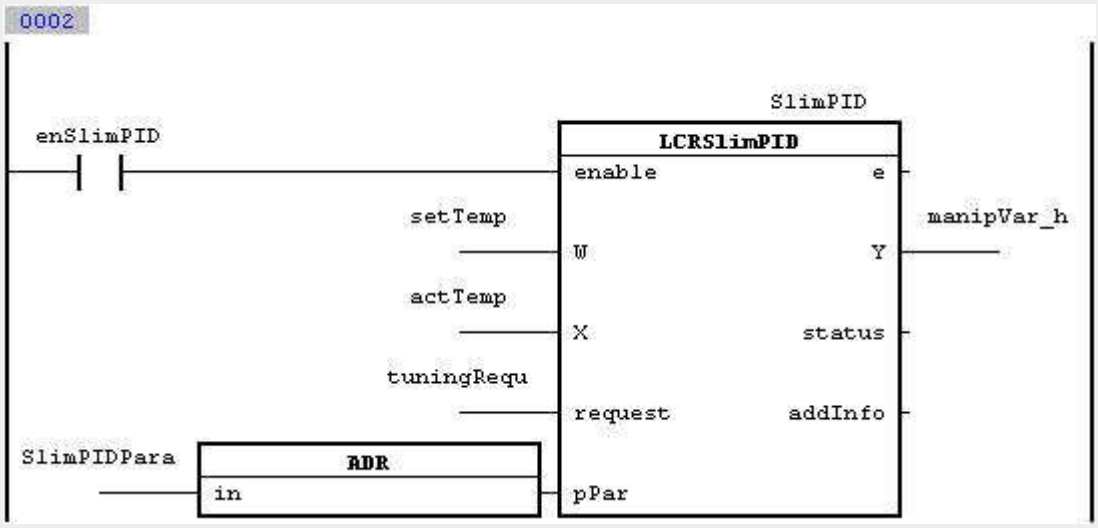


Fig. 54: LCRSlimPID() function block

Variable declaration:

Name	Type	Scope	Attribute
SimModExt	LCRSimModExt	local	memory
SlimPID	LCRSlimPID	local	memory
SlimPIDPara	lcrslimpid_par_typ	local	memory
actTemp	REAL	local	memory
enSimModExt	BOOL	local	memory
enSlimPID	BOOL	local	memory
manipVar_h	REAL	local	memory
setTemp	REAL	local	memory
tuningRequ	UDINT	local	memory

Fig. 55: Variable declaration

The output value *Y* of the function block LCRSlimPID() is copied to the *manipVar_h* variable and forms the manipulate variable that's fed to the LCRSimModExt() function block as heating control action at the *Alpha_h* input. The resulting controlled variable *y* is copied to the *actTemp* variable and fed to the LCRSlimPID() function block as the actual value at input *X*.

A closed control loop results in this way.

Evaluating the traces:

First, we will try to find an approximate setting for the gain.

For our first attempt we have selected three different gains:

- kp= 0.5
- kp= 3.0
- kp= 8.0

and recorded the set and actual temperature and the gain.

kp= 0.5: The oscillation fades quickly, the controlled variable quiets down quickly, and the remaining controller deviation is very large.

kp= 3.0: The oscillation fades in an acceptable time, and the remaining control deviation is less than 10%.

kp= 8.0: The oscillation keeps going, and the control loop is unstable.

From this experiment, kp = 3.0 would be selected as the most suitable gain.

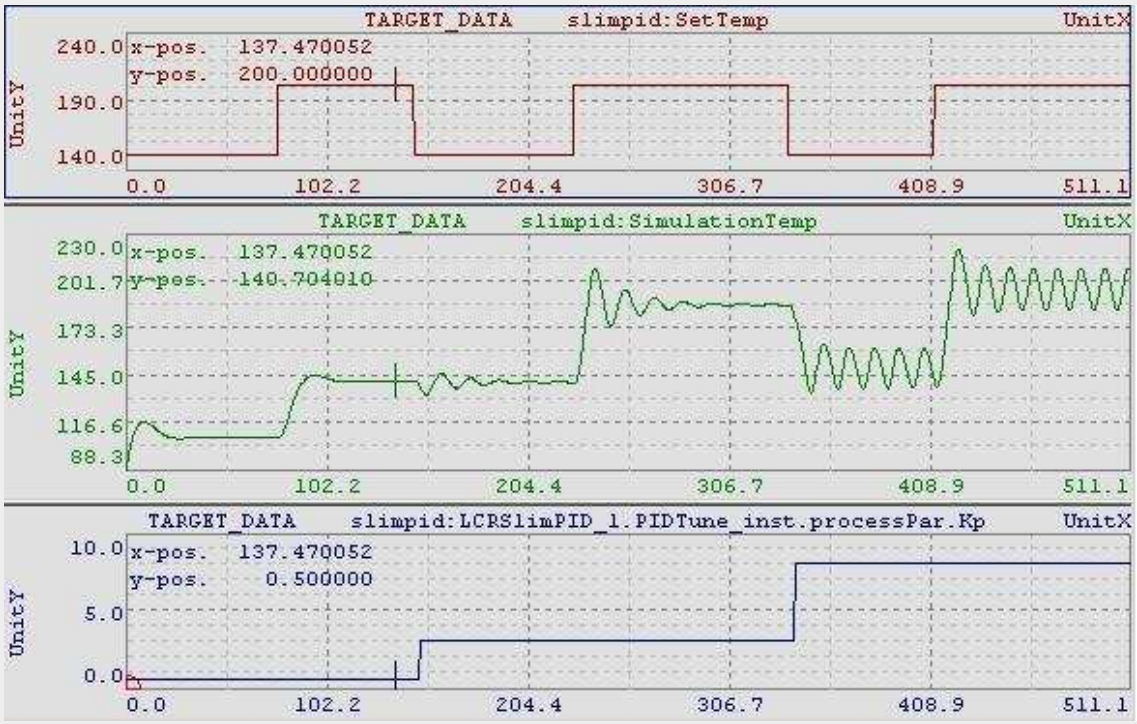


Fig. 56: Approximate gain settings for the LCRSlimPID() function block

If the oscillation fades faster when the set value changes, then the gain can be reduced a bit.

Another experiment allows us to examine the behavior when the gain is lower:

- kp= 1
- kp= 2
- kp= 3

The trace shows the following connection:

The higher the gain kp, the smaller the remaining controller deviation. However, the control loop becomes more and more unstable as kp increases (the oscillations after the set value is changed fade more and more slowly).

Depending on the requirements, a gain between 2 and 3 would be selected from this experiment.

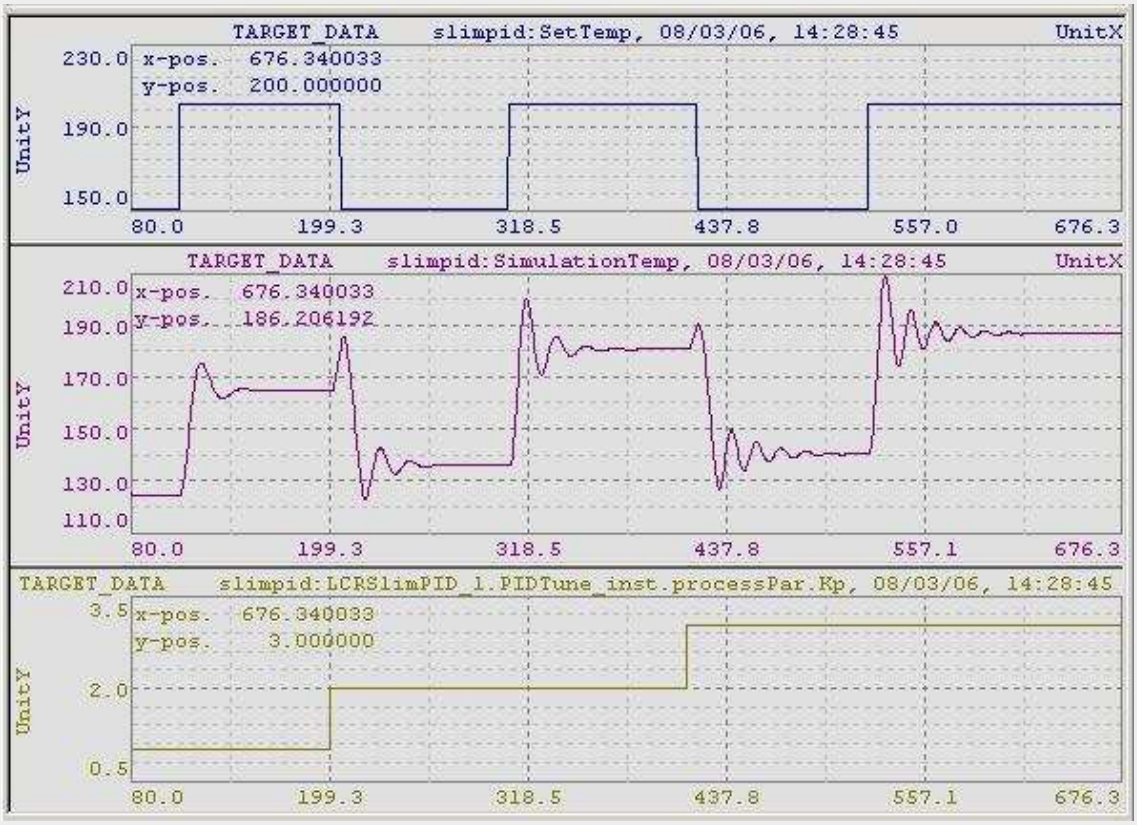


Fig. 57: Fine tuning gain for the LCRSlimPID() function block

15.2.2 Task: LCRSlimPID() controller settings according to Ziegler/Nichols

k_{crit} and T_{crit} must first be determined before they can used in the table for calculating the controller parameters.

The following example gains are shown here in Trace:

- kp= 3.0
- kp= 4.0
- kp= 5.0

You can see that, beginning at a gain k_{crit} = 5.0 sets a continuous oscillation with constant amplitude and period length of T_{crit} = 12 seconds.

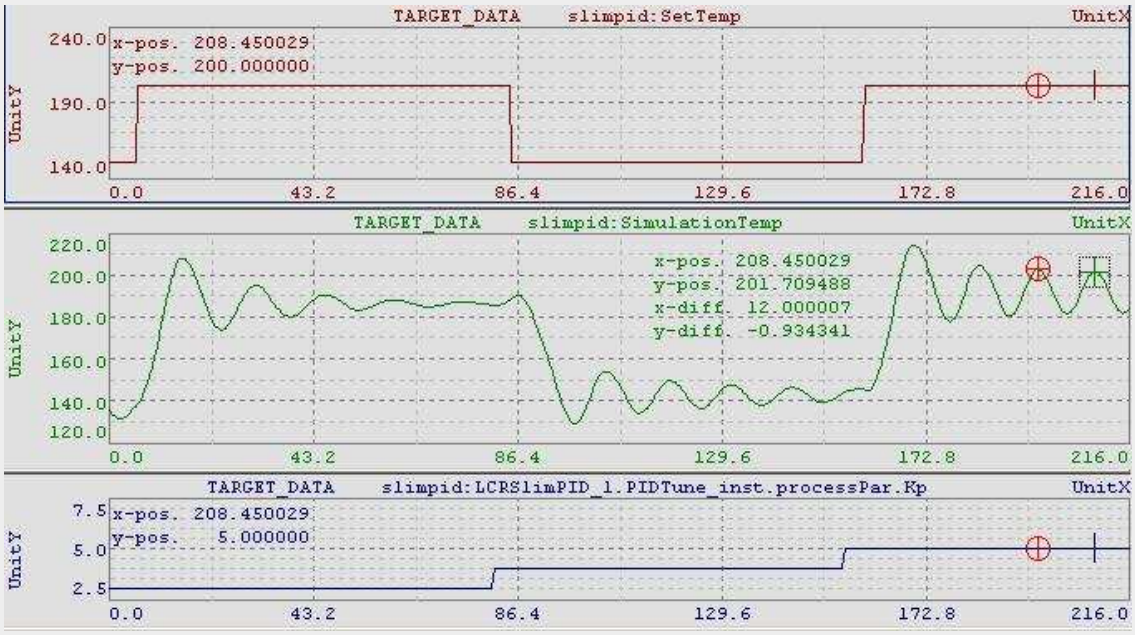


Fig. 58: Trace P-element instability – Ziegler/Nichols

Now both parameters only need to be used in the Ziegler/Nichols calculation table.

	Control parameters		
Controller type	**kp**	**Tn**	**Tv**
P	2.5		
PI	2.25	10.2	
PID	3.0	6.0	1.44

We have now configured our PID controller according to the Ziegler/Nichols method using the following values:

- kp= 3.0
- Tn = 6.0 s
- Tv = 1.44 s

As you can see in the Trace below, the temperature control has become much faster and more stable.

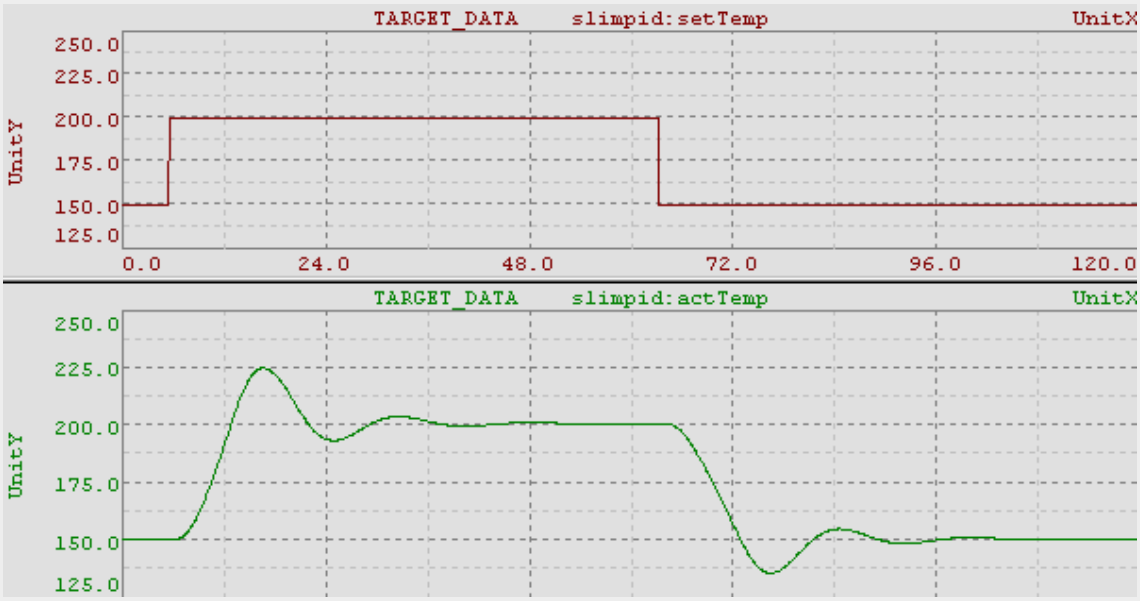


Fig. 59: PID controller configured with the Ziegler/Nichols method

15.2.3 Task: LCRSlimPID() auto-tuning with oscillation attempt

Before tuning can be started with oscillation, a set value must be specified that is close to the later working point. 150 °C is preset as the set value in this example.

To start standard tuning with oscillation, the *request* input of the LCRSlimPID() function block must be set to LCRSLIMPID_REQU_OSCILLATE (1). Only when tuning has finished (after changing to normal controlled operation) can *request* be set back to LCRSLIMPID_REQU_OFF (0).

Trace of the tuning with subsequent automatic activation of the controller:

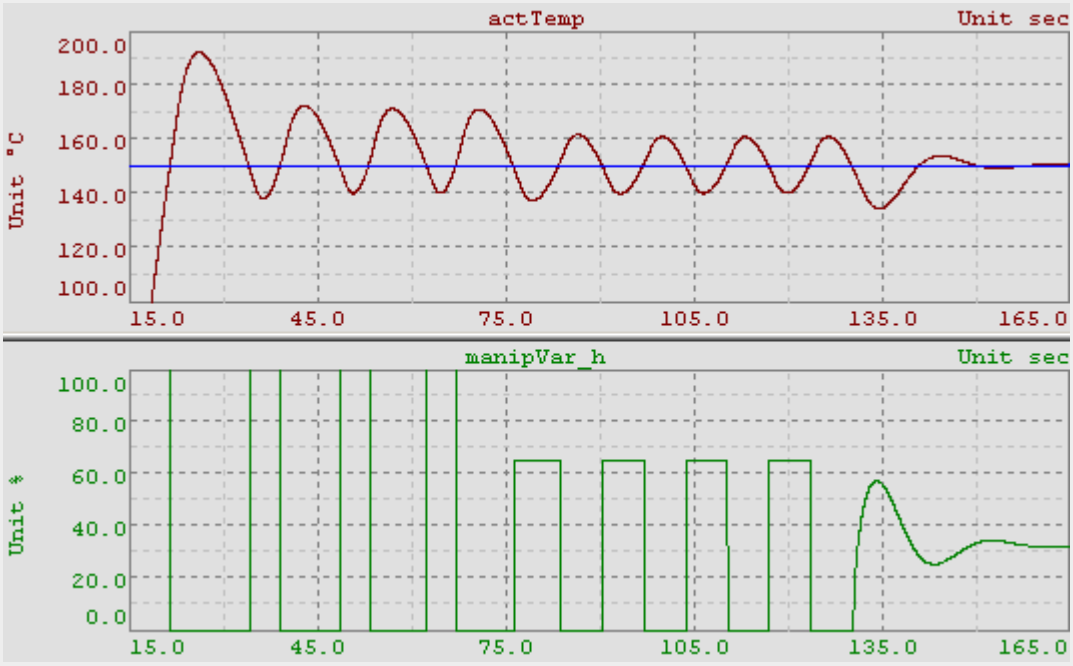


Fig. 60: Trace LCRSlimPID() tuning with step response

Step response controller parameters:

Name	Type	Scope	Value
tuningRequ	UDINT	local	4
SlimPIDPara	lcrslimpid_par_typ	local	
Y_max	REAL		100.0
Y_min	REAL		0.0
Kp	REAL		2.37954
Tn	REAL		6.61889
Tv	REAL		1.65472
Kfbk	REAL		0.0

Fig. 61: LCRSlimPID() tuning parameters oscillation attempt

A constant manipulated variable (*Y*) is set after the settling phase is complete. This is used later for the step response.

Name	Type	Scope	Value
SlimPID	LCRSlimPID	local	
enable	BOOL		TRUE
W	REAL		150.0
X	REAL		150.0
request	UDINT		1
pPar	UDINT		63726096
e	REAL		0.0
Y	REAL		32.0513
status	UINT		0
addInfo	UINT		0

Fig. 62: LCRSlimPID() constant manipulated variable

15.2.4 Task: LCRSlimPID() auto-tuning with step response

Before tuning can be started with step response, a set value must be specified that is close to the later working point. 150 °C is preset as the set value in this example. Furthermore, the necessary manipulated variable must also specified for the set value. In this case, we already know that the controlled system requires a constant manipulated variable of approximately 32% when in a steady state. Therefore, the internal variable *Y0* is set to 32 and Y1 to 45 (approximately 30% larger than *Y0*) in the watch window.

To start standard tuning with step response, the *request* input of the LCRSlimPID() function block must be set to LCRSLIMPID_REQU_STEPRESPONSE (2). Only when tuning has finished (after changing to normal controlled operation) can *request* be set back to LCRSLIMPID_REQU_OFF (0).

Trace of the tuning with subsequent automatic activation of the controller:

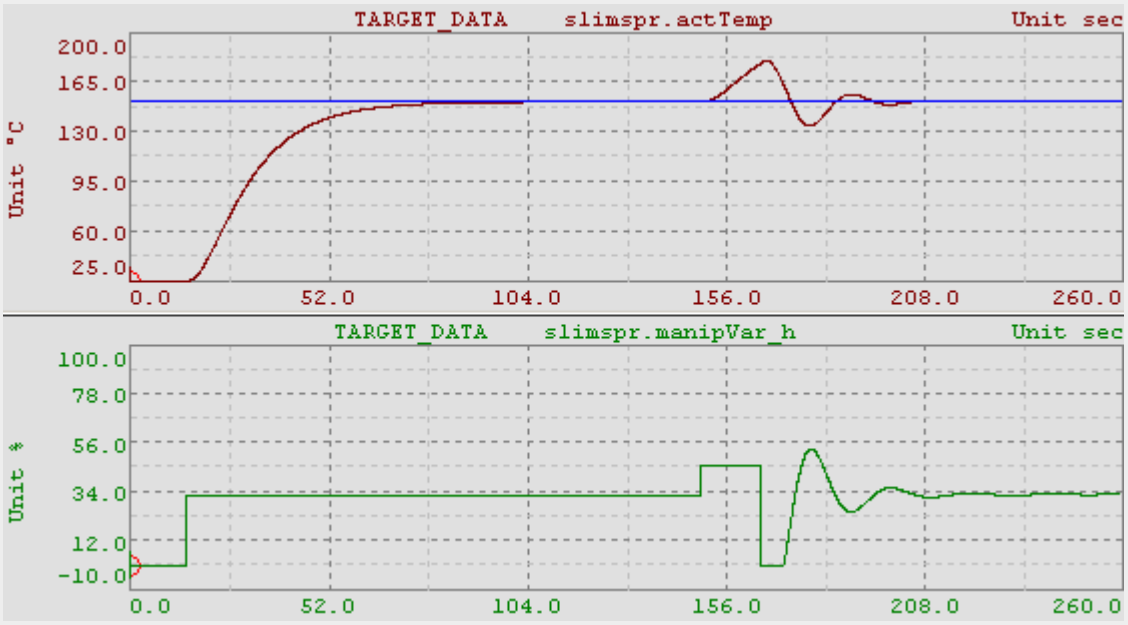


Fig. 63: Trace LCRSlimPID() tuning step response disturbance variable design

To start a tuning procedure with step response, for which non-periodic reference parameters must be determined, the *request* input of the LCRSlimPID() function block is set to 4112 (see online help) . *request* is only reset to LCRSLIMPID_REQU_OFF (0) once the tuning procedure is complete. Just like before, the internal variable *Y0* is set to 32 and *Y1* to 45.

Trace of the tuning with subsequent automatic activation of the controller:

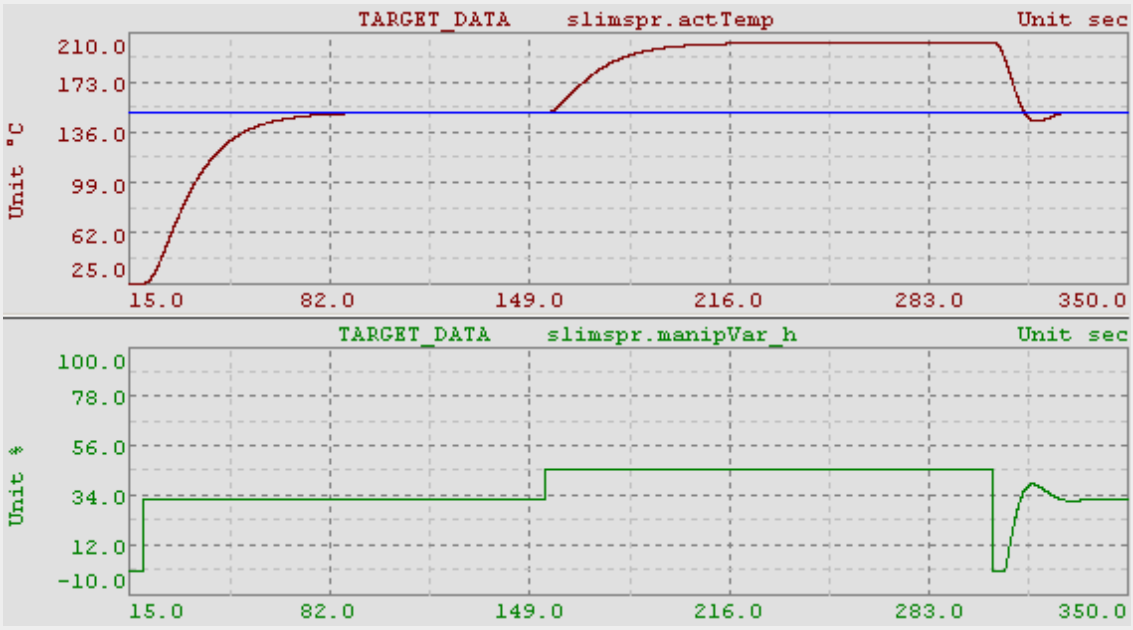


Fig. 64: Trace LCRSlimPID() tuning step response reference variable design

Control parameters determined during the standard step response (non-periodic disturbance variable design, LCRSLIMPID_REQU_STEPRESPONSE (2)):

Name	Type	Scope	Value
SlimPIDPara	lcrslimpid_par_typ	local	
Y_max	REAL		100.0
Y_min	REAL		0.0
Kp	REAL		1.83915
Tn	REAL		7.31492
Tv	REAL		1.28011
Kfbk	REAL		0.0

Fig. 65: LCRSlimPID() PID parameters disturbance variable design

During auto-tuning with step response to determine non-periodic reference parameters (*request* = 4112):

Name	Type	Scope	Value
SlimPIDPara	lcrslimpid_par_typ	local	
Y_max	REAL		100.0
Y_min	REAL		0.0
Kp	REAL		1.16157
Tn	REAL		23.014
Tv	REAL		1.52394
Kfbk	REAL		0.0

Fig. 66: LCRSlimPID() PID parameters reference variable design

The PID parameters determined with the disturbance design are considerably more aggressive than those from the reference design. The reference design should be used if controller behavior is desired which reaches the set value with the least amount of overshoot. The disturbance design should be used if the control loop should quickly adjust for disturbances and overshoots are not of major importance.

15.2.5 Task: Control a temperature system (heating and cooling) using LCRTempPID()

Ladder diagram: Initialization routine

The PID parameters are transferred to the structure connected to the *pSettings* input, cooling is enabled, the delay time for the set value implementation is set to 0.1s and the set value is set to 180°C.

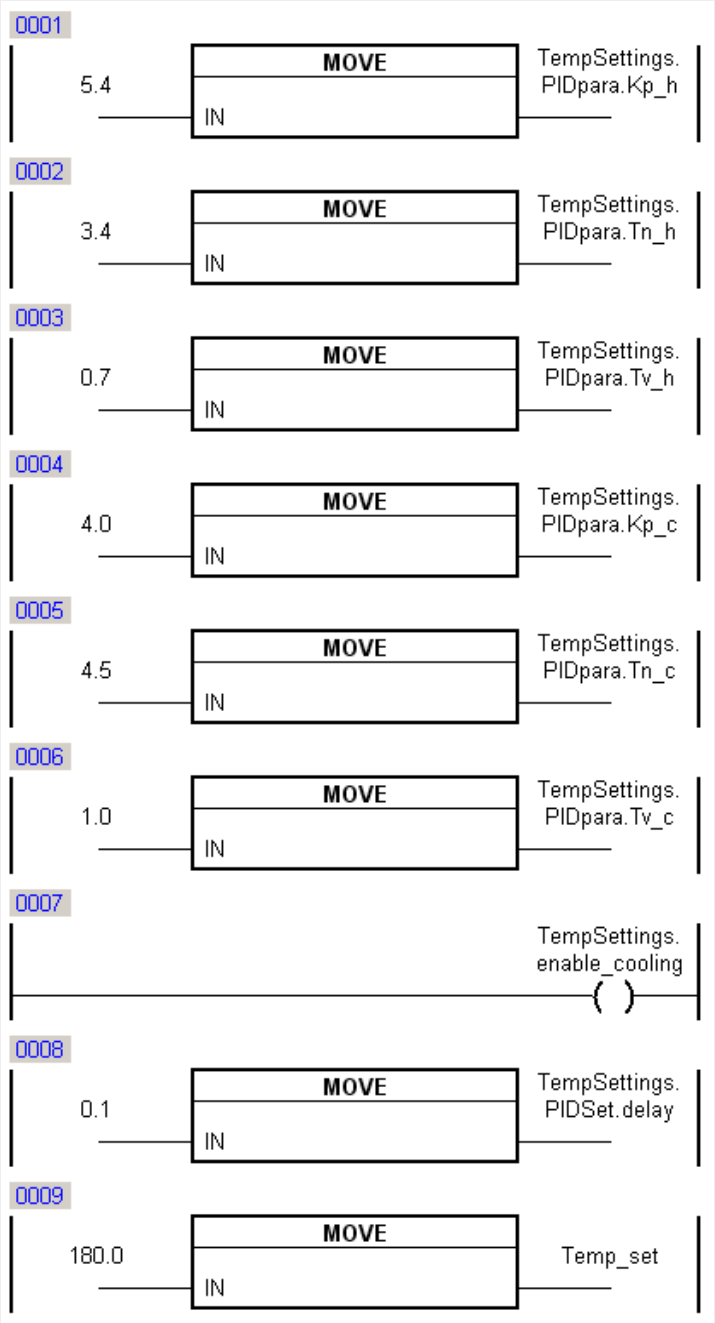


Fig. 67:LCRTempPID() initializations routine

Regulating a temperature system with opposing manipulated variables (heating and cooling) using the LCRTempPID() function block.

Ladder diagram: LCRTempPID()

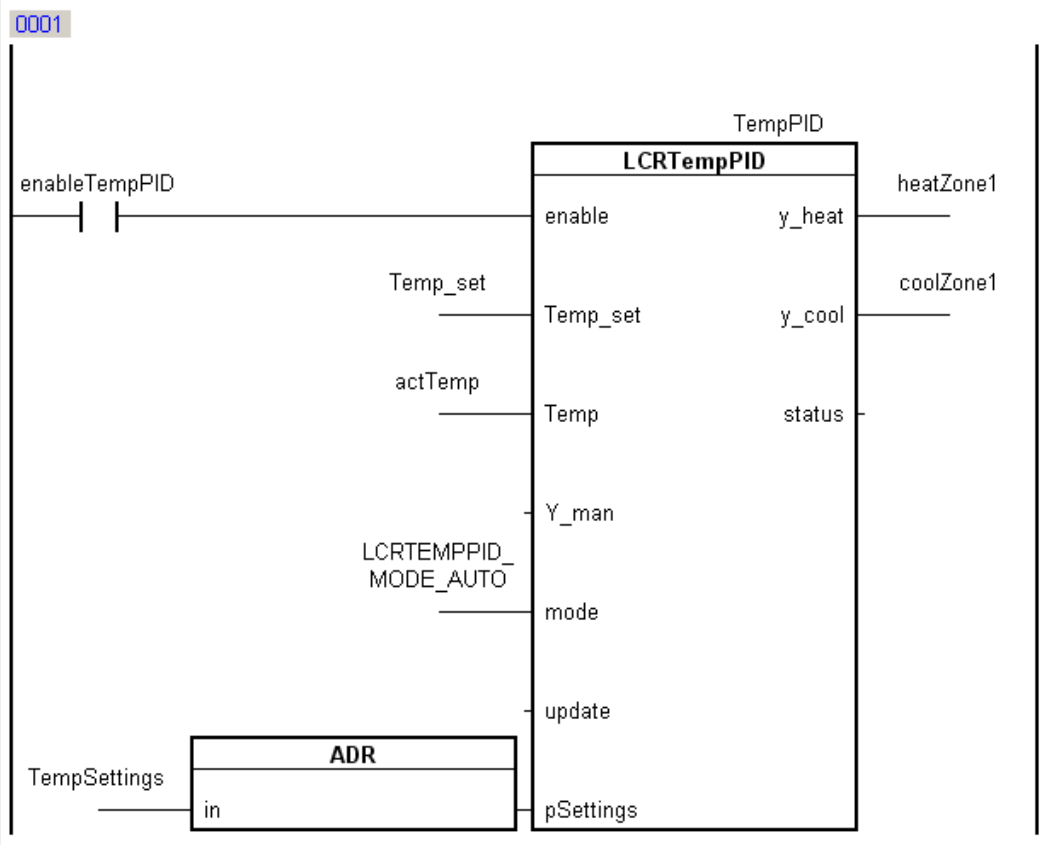


Fig. 68: Function block LCRTempPID()

Variable declaration:

Name	Type	Scope	Attribute	Value
LCRTEMPPID_M	UDINT	global	constant	1
TempPID	LCRTempPID	local	memory	
TempSettings	lcrtemp_set_typ	local	memory	
Temp_set	REAL	local	memory	
actTemp	REAL	local	memory	
coolZone1	REAL	local	memory	
enableTempPID	BOOL	local	memory	
heatZone1	REAL	local	memory	
zone1	LCRSimModExt	local	memory	

Fig. 69: LCRTempPID() variable declaration

Ladder diagram: LCRSimModExt()

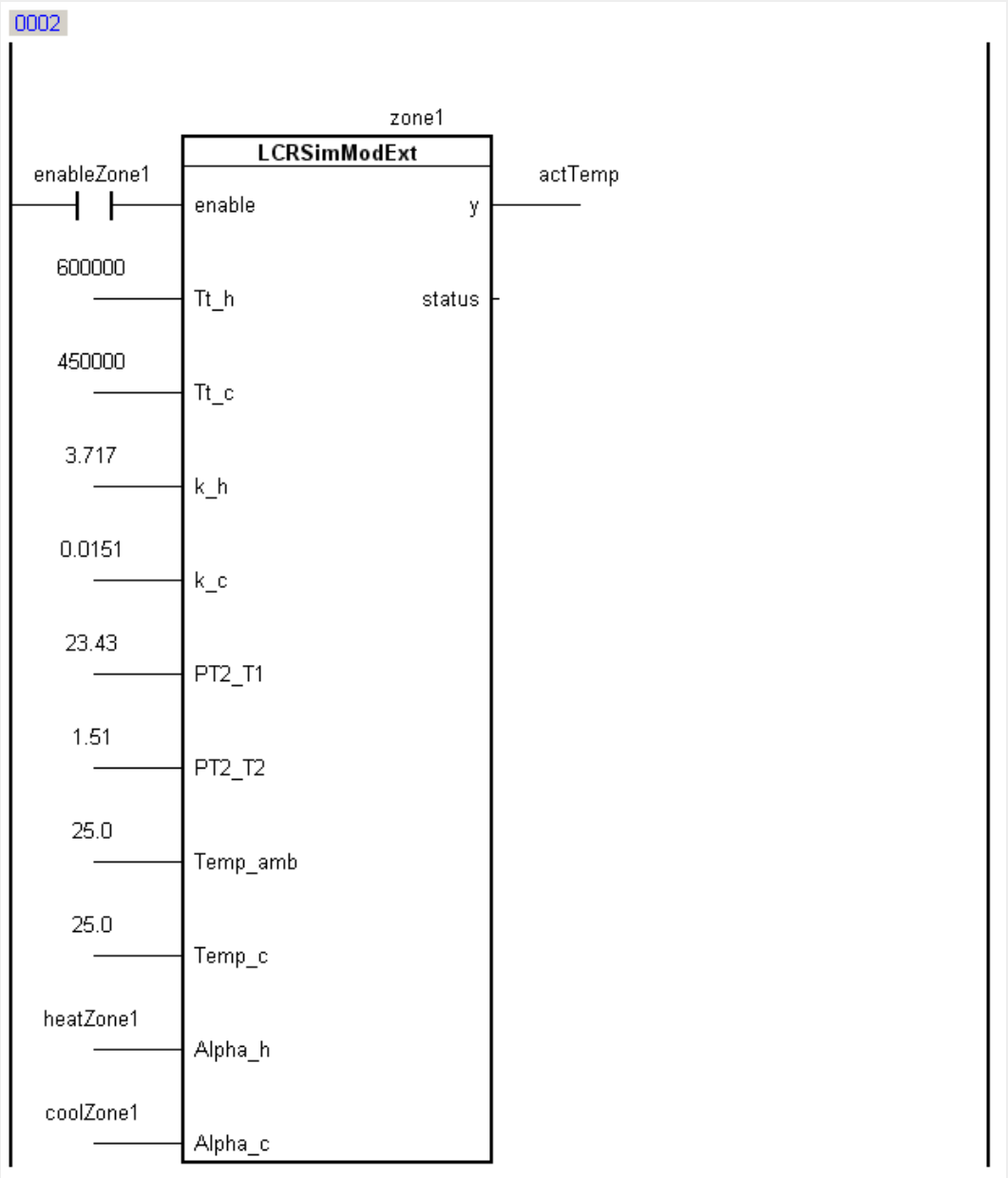


Fig. 70: LCRSimModExt() Function block

Step responses:

Description	Temperature
Output temperature	25°C
1. Jump	180°C
2. Jump	183°C
3. Jump	200°C
4. Jump	170°C

Trace:

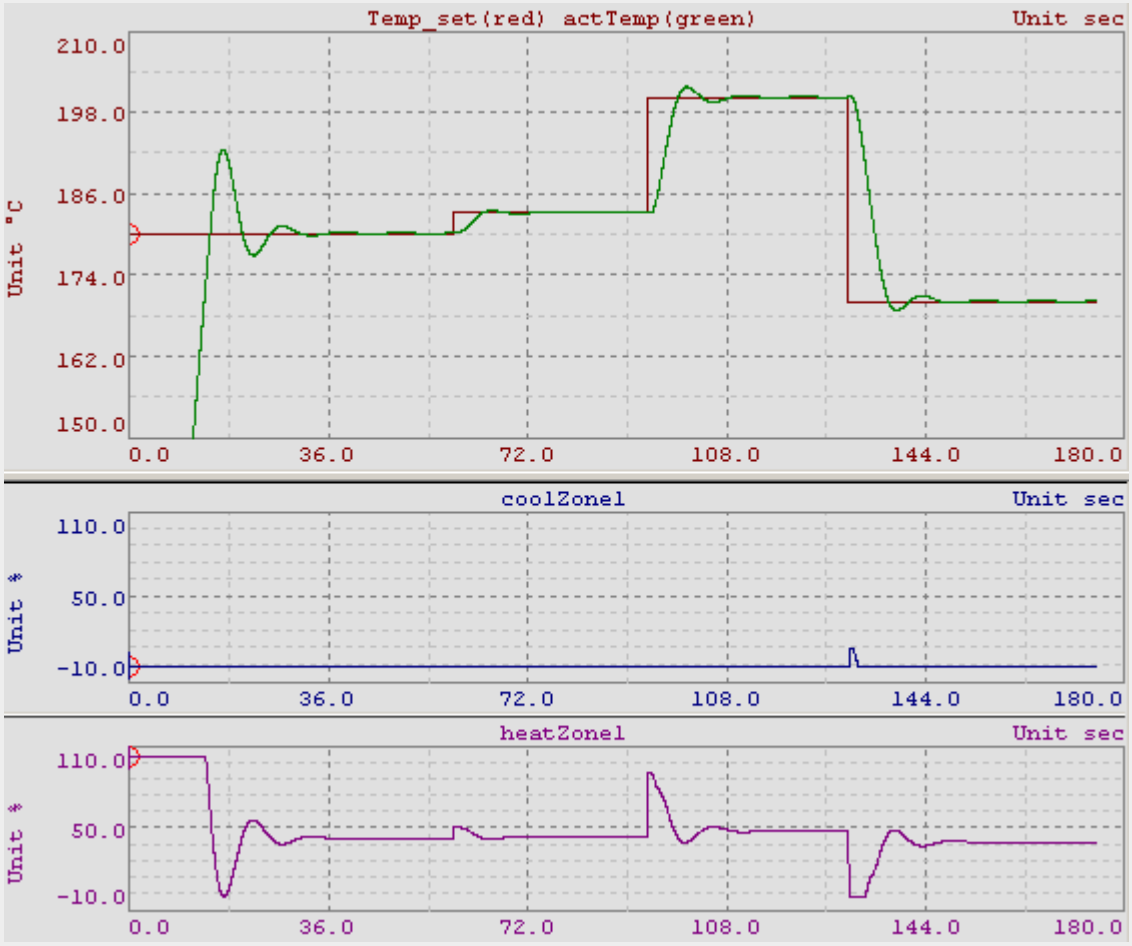


Fig. 71: LCRTempPID() set value jumps

15.2.6 Task: Tuning a temperature system using LCRTempTune()

Ladder diagram: Initialization routine

All time-critical variables (delays, gradients and filter times) are chosen smaller than the default value because the simulation runs faster than on a real extruder. The tuning procedure for cooling is also enabled.

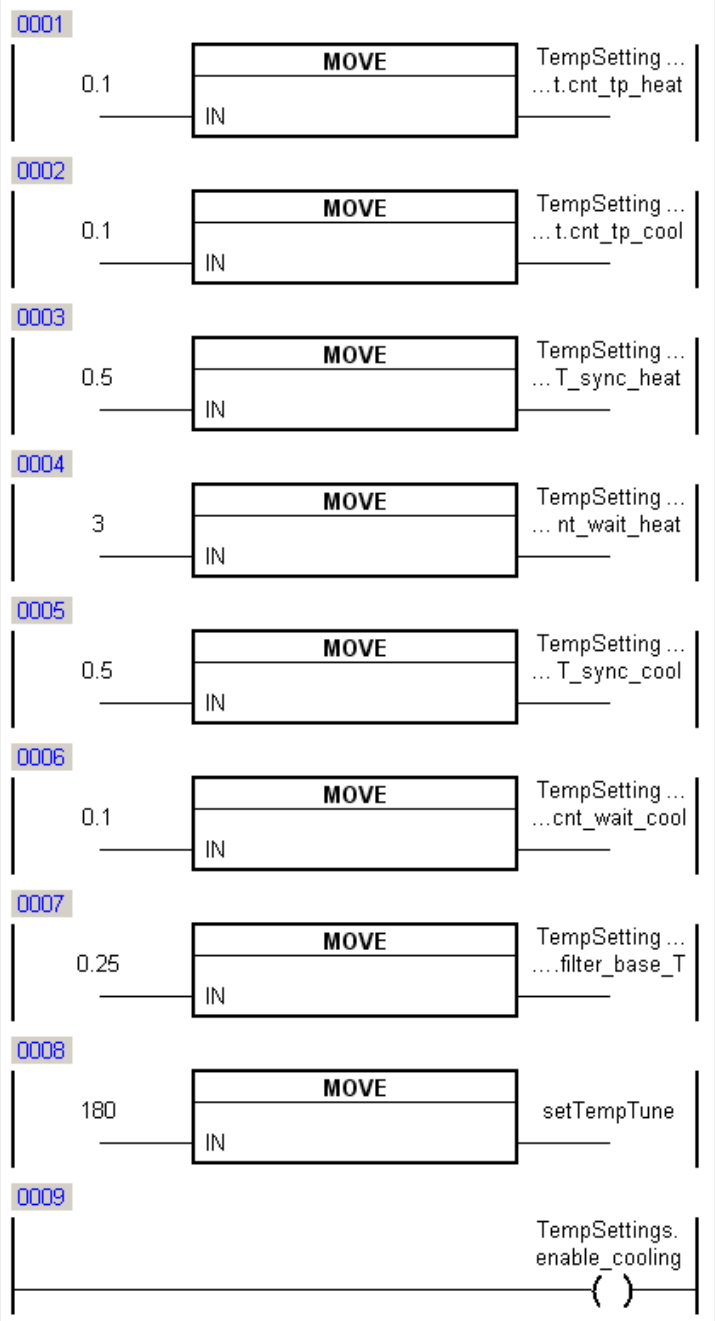


Fig. 72: LCRTempTune() initialization routine

Ladder diagram: LCRTempTune()

Implementing the tuning procedure using the LCRTempTune() function block. The "*rdyTo*" outputs are linked with the "*okTo*" inputs.

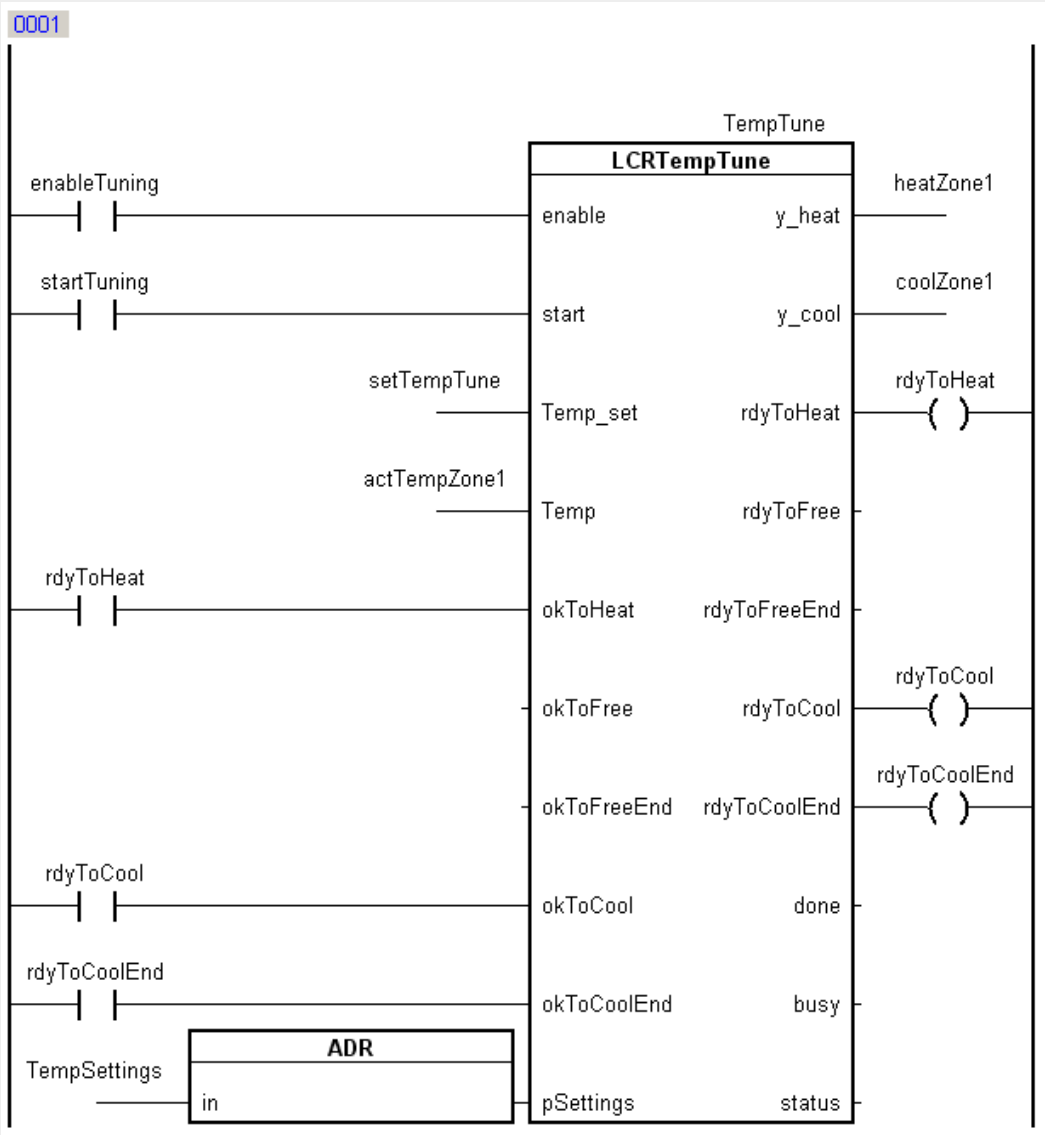


Fig. 73: Function block LCRTempTune()

Ladder diagram: LCRSimModExt()

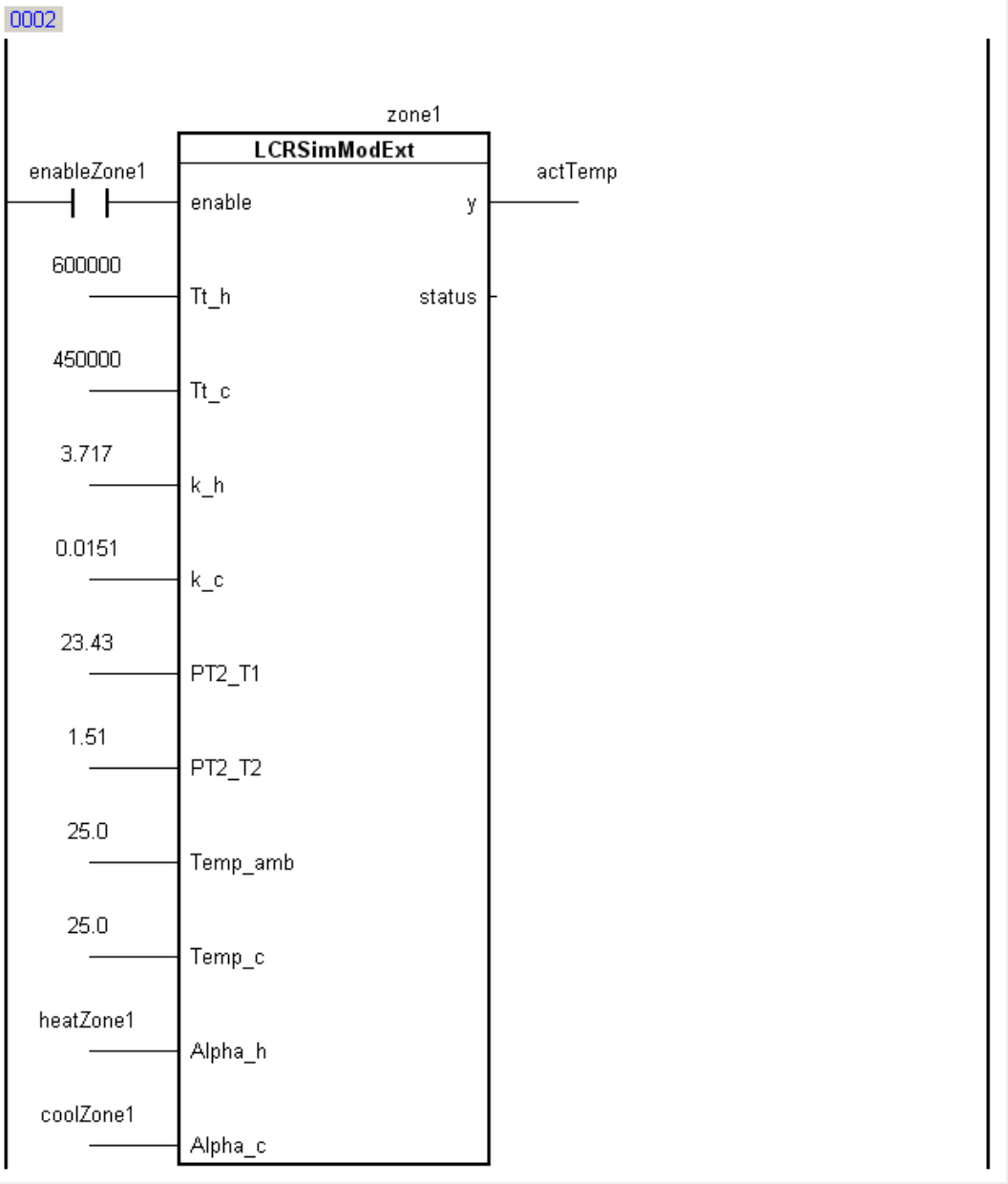


Fig. 74: LCRSimModExt() Function block

Variable declaration:

Name	Type	Scope	Attribute
TempSettings	lcrtemp_set_typ	local	memory
TempTune	LCRTempTune	local	memory
actTempZone1	REAL	local	memory
coolZone1	REAL	local	memory
enableTuning	BOOL	local	memory
enableZone1	BOOL	local	memory
heatZone1	REAL	local	memory
rdyToCool	BOOL	local	memory
rdyToCoolEnd	BOOL	local	memory
rdyToHeat	BOOL	local	memory
setTempTune	REAL	local	memory
startTuning	BOOL	local	memory
zone1	LCRSimModExt	local	memory

Fig. 75: LCRTempTune() variable declaration

Trace:

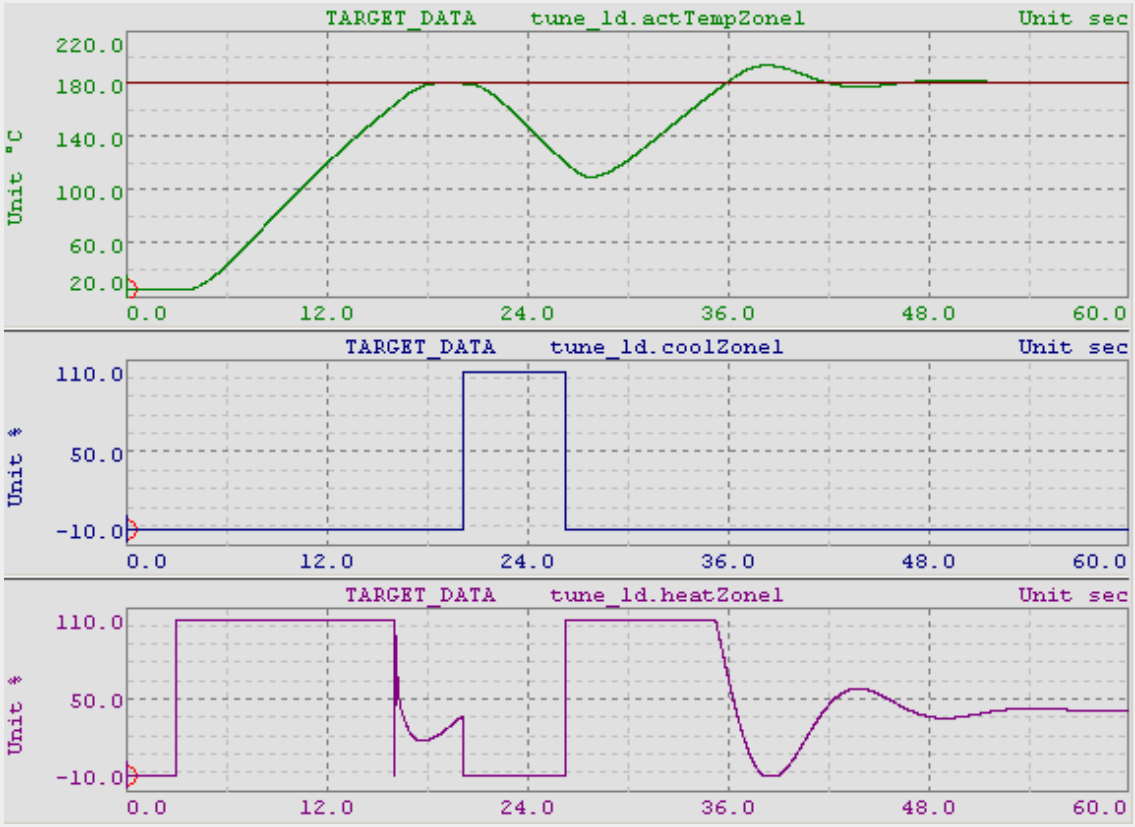


Fig. 76: Trace LCRTempTune() Tuning

Determined PID parameters:

Name	Type	Scope	Force	Value
TempSettings	lcrtemp_set_typ	local		
enable_cooling	BOOL			TRUE
PIDpara	lcrtemp_pid_opt_typ			
Kp_h	REAL			5.4577
Tn_h	REAL			3.34844
Tv_h	REAL			0.703172
Kp_c	REAL			4.48779
Tn_c	REAL			4.15687
Tv_c	REAL			0.872942

Fig. 77: LCRTempTune() PID parameters

15.2.7 Task: Synchronous tuning of two temperature systems using the function blocks LCRTempTune() and LCRTempPID()

Implementation of both control loops in Structured Text.

```
(* ------> init program *)
(* init user settings tuning zone 1 *)
LCRTempSet1.enable_cooling              := TRUE ; (* cool tuning
                                                    enabled *)
(* do not use default parameters for example due to smaller time
   constants *)
LCRTempSet1.TuneSet.cnt_tp_heat         := 0.1;  (* seconds *)
LCRTempSet1.TuneSet.cnt_tp_cool         := 0.1;  (* seconds *)
LCRTempSet1.TuneSet.delta_dT_sync_heat := 0.5;  (* °C/sec *)
LCRTempSet1.TuneSet.cnt_wait_heat       := 3;    (* seconds *)
LCRTempSet1.TuneSet.delta_dT_sync_cool := 0.5;  (* °C/sec *)
LCRTempSet1.TuneSet.cnt_wait_cool       := 1;    (* seconds *)
LCRTempSet1.TuneSet.filter_base_T       := 0.25; (* seconds *)
LCRTempTune1.pSettings                  := ADR(LCRTempSet1);
(* end init user settings tuning zone 1 *)

(*  init user settings tuning zone 2*)
LCRTempSet2.enable_cooling              := TRUE ; (* cool tuning
                                                    enabled *)
(* do not use default parameter for example due to smaller time
   constants *)
LCRTempSet2.TuneSet.cnt_tp_heat         := 0.1;  (* seconds *)
LCRTempSet2.TuneSet.cnt_tp_cool         := 0.1;  (* seconds *)
LCRTempSet2.TuneSet.delta_dT_sync_heat := 0.5;  (* °C/sec *)
LCRTempSet2.TuneSet.cnt_wait_heat       := 3;    (* seconds *)
LCRTempSet2.TuneSet.delta_dT_sync_cool := 0.5;  (* °C/sec *)
LCRTempSet2.TuneSet.cnt_wait_cool       := 1;    (* seconds *)
LCRTempSet2.TuneSet.filter_base_T       := 0.25; (* seconds *)
LCRTempTune2.pSettings                  := ADR(LCRTempSet2);
(* end init user settings tuning zone 2 *)

(* init user settings LCRTempPID1 *)
(* wait 0.1 seconds before switching to new set temperature *)
LCRTempSet1.PIDSet.delay      := 0.1;           (* seconds *)
LCRTempPID1.pSettings         := ADR(LCRTempSet1);
(* end init user settings LCRTempPID1 *)

(* init user settings LCRTempPID2 *)
(* wait 0.1 seconds before switching to new set temperature *)
LCRTempSet2.PIDSet.delay      := 0.1;           (* seconds *)
LCRTempPID2.pSettings         := ADR(LCRTempSet2);
(* end init user settings LCRTempPID2 *)

(* parameters for simulated extruder zone 1 *)
zone1.enable    := TRUE;
zone1.Tt_h      := 600000;                      (* microseconds *)
zone1.Tt_c      := 450000;                      (* microseconds *)
zone1.k_h       := 3.717;
zone1.k_c       := 0.0151;
zone1.PT2_T1    := 23.43;
zone1.PT2_T2    := 1.51;
zone1.Temp_amb  := 25.0;                        (* °C *)
zone1.Temp_c    := 25.0;                        (* °C *)
(* end parameters for simulated extruder zone 1 *)
```

```
(* parameters for simulated extruder zone 2 *)
zone2.enable    := TRUE;
zone2.Tt_h      := 1200000;         (* microseconds *)
zone2.Tt_c      := 800000;          (* microseconds *)
zone2.k_h       := 4;
zone2.k_c       := 0.013;
zone2.PT2_T1    := 28;
zone2.PT2_T2    := 1.8;
zone2.Temp_amb  := 25.0;            (* °C *)
zone2.Temp_c    := 25.0;            (* °C *)
(* end parameters for simulated extruder zone 2 *)

(* set tuning temperatures and set temperatures after tuning*)
setTempTune := 180;                 (* °C *)
setTempPID1 := 180;                 (* °C *)
setTempPID2 := 180;                 (* °C *)

(* ------> cyclic program *)
(* DESCRIPTION: start autotuning with enableTuning = TRUE and
   startTuning = TRUE *)

(* read out current temperatures *)
actTempZone1 := zone1.y;
actTempZone2 := zone2.y;

(* enable tuning function blocks *)
LCRTempTune1.enable := enableTuning;
LCRTempTune2.enable := enableTuning;

(* start autotuning *)
LCRTempTune1.start := startTuning;
LCRTempTune2.start := startTuning;

(* current set temperature for tuning function block*)
LCRTempTune1.Temp_set := setTempTune;
LCRTempTune2.Temp_set := setTempTune;

(* current temperature for tuning function block *)
LCRTempTune1.Temp := actTempZone1;
LCRTempTune2.Temp := actTempZone2;

(* synchronisation of autotuning *)
LCRTempTune1.okToHeat     := (LCRTempTune1.rdyToHeat AND
LCRTempTune2.rdyToHeat);
LCRTempTune2.okToHeat     := (LCRTempTune1.rdyToHeat AND
LCRTempTune2.rdyToHeat);
LCRTempTune1.okToCool     := (LCRTempTune1.rdyToCool AND
LCRTempTune2.rdyToCool);
LCRTempTune2.okToCool     := (LCRTempTune1.rdyToCool AND
LCRTempTune2.rdyToCool);
LCRTempTune1.okToCoolEnd  := (LCRTempTune1.rdyToCoolEnd AND
LCRTempTune2.rdyToCoolEnd);
LCRTempTune2.okToCoolEnd  := (LCRTempTune1.rdyToCoolEnd AND
LCRTempTune2.rdyToCoolEnd);

(* call tuning function blocks LCRTempTune() *)
LCRTempTune1();
LCRTempTune2();
```

```
(* when tuning has finished switch to LCRTempPID() controllers *)
IF ((LCRTempTune1.done AND LCRTempTune2.done) OR TempPID_enable) THEN
   (* enable controller function blocks (auto mode) *)
   LCRTempPID1.enable := TRUE ;
   LCRTempPID1.mode   := LCRTEMPPID_MODE_AUTO;
   LCRTempPID2.enable := TRUE ;
   LCRTempPID2.mode   := LCRTEMPPID_MODE_AUTO;

   (* current set temperature for controller function blocks *)
   LCRTempPID1.Temp_set := setTempPID1;
   LCRTempPID2.Temp_set := setTempPID2;

   (* current temperature for controller function blocks *)
   LCRTempPID1.Temp := actTempZone1;
   LCRTempPID2.Temp := actTempZone2;

   (* manipulated variable from controller function blocks *)
   heatZone1       := LCRTempPID1.y_heat;
   coolZone1       := LCRTempPID1.y_cool;
   heatZone2       := LCRTempPID2.y_heat;
   coolZone2       := LCRTempPID2.y_cool;

   (* disable Tunings and start controllers *)
   enableTuning    := FALSE;
   startTuning     := FALSE;
   TempPID_enable  := TRUE;
ELSE
   (*manipulated variable from tuning function blocks *)
   heatZone1 := LCRTempTune1.y_heat;
   coolZone1 := LCRTempTune1.y_cool;
   heatZone2 := LCRTempTune2.y_heat;
   coolZone2 := LCRTempTune2.y_cool;

   (*disable LCRTempPIDs so no doubleacting is possible*)
   LCRTempPID1.enable := FALSE;
   LCRTempPID2.enable := FALSE;
END_IF

(* call controller function blocks LCRTempPID() *)
LCRTempPID1();
LCRTempPID2();

(* call simulated extruder zones and handover the manipulated
   variables *)
zone1.Alpha_h := heatZone1;
zone1.Alpha_c := coolZone1;
zone1();
zone2.Alpha_h := heatZone2;
zone2.Alpha_c := coolZone2;
zone2();
```

Variable declaration:

Name	Type	Scope	Attribute	Value
LCRTEMPPID_MODI	UDINT	global	constant	1
LCRTempPID1	LCRTempPID	local	memory	
LCRTempPID2	LCRTempPID	local	memory	
LCRTempSet1	lcrtemp_set_typ	local	memory	
LCRTempSet2	lcrtemp_set_typ	local	memory	
LCRTempTune1	LCRTempTune	local	memory	
LCRTempTune2	LCRTempTune	local	memory	
TempPID_enable	BOOL	local	memory	
actTempZone1	REAL	local	memory	
actTempZone2	REAL	local	memory	
coolZone1	REAL	local	memory	
coolZone2	REAL	local	memory	
enableTuning	BOOL	local	memory	
heatZone1	REAL	local	memory	
heatZone2	REAL	local	memory	
setTempPID1	REAL	local	memory	
setTempPID2	REAL	local	memory	
setTempTune	REAL	local	memory	
startTuning	BOOL	local	memory	
zone1	LCRSimModExt	local	memory	
zone2	LCRSimModExt	local	memory	

Fig. 78: LCRTempPID()+LCRTempTune() variable declaration

Determined PID parameters:

Name	Type	Scope	Force	Value
LCRTempSet2	lcrtemp_set_typ	local		
PIDpara	lcrtemp_pid_opt_typ			
Kp_h	REAL			4.09106
Tn_h	REAL			4.95874
Tv_h	REAL			1.04134
Kp_c	REAL			3.28678
Tn_c	REAL			7.36692
Tv_c	REAL			1.54705
LCRTempSet1	lcrtemp_set_typ	local		
PIDpara	lcrtemp_pid_opt_typ			
Kp_h	REAL			5.4577
Tn_h	REAL			3.34844
Tv_h	REAL			0.703172
Kp_c	REAL			3.94967
Tn_c	REAL			4.65961
Tv_c	REAL			0.978518

Fig. 79: LCRTempPID()+LCRTempTune() PID parameters

Trace:

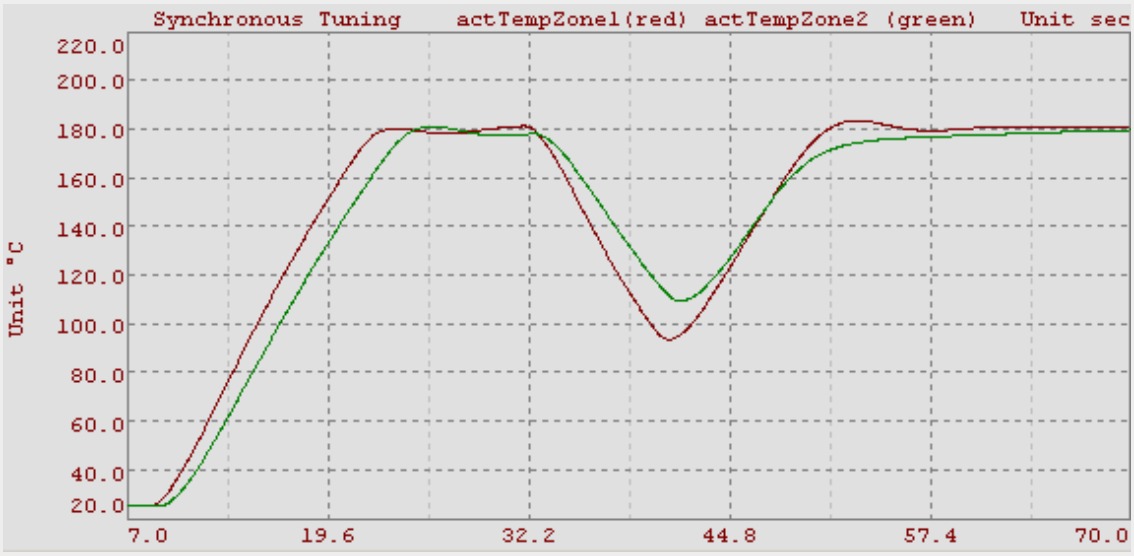


Fig. 80: LCRTempPID()+LCRTempTune() Tuning

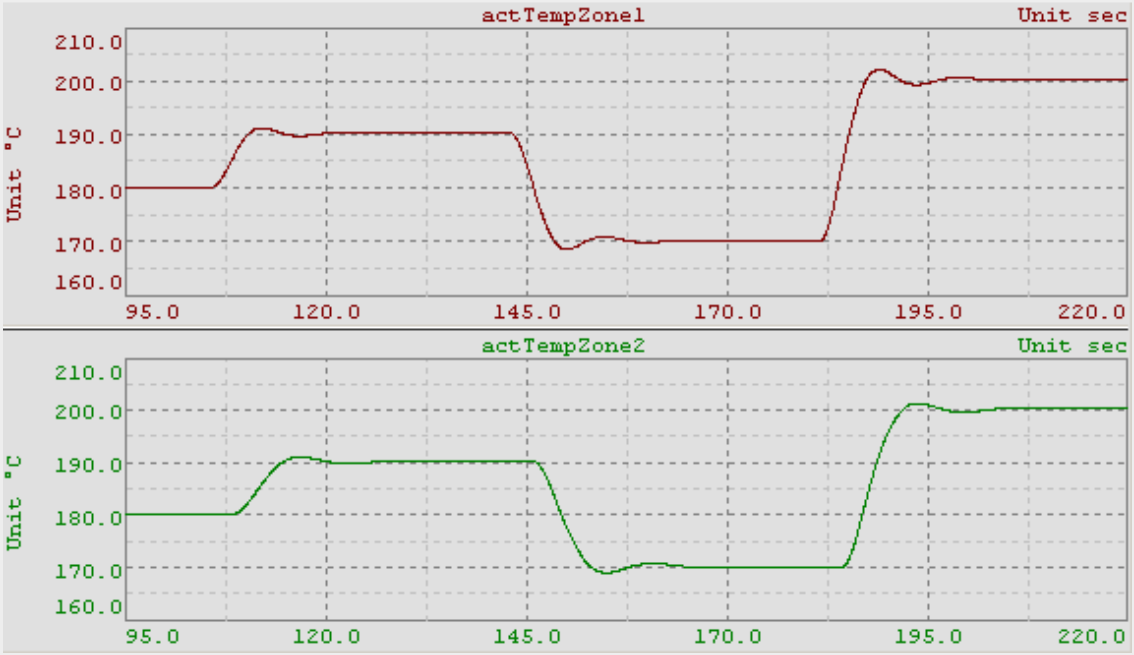


Fig. 81: LCRTempPID()+LCRTempTune() set value jumps

15.2.8 Task: B&R simulation model 4SIM.00-01

Ladder diagram: Initialization routine

The default values are replaced by suitable values because they are not feasible for this system.

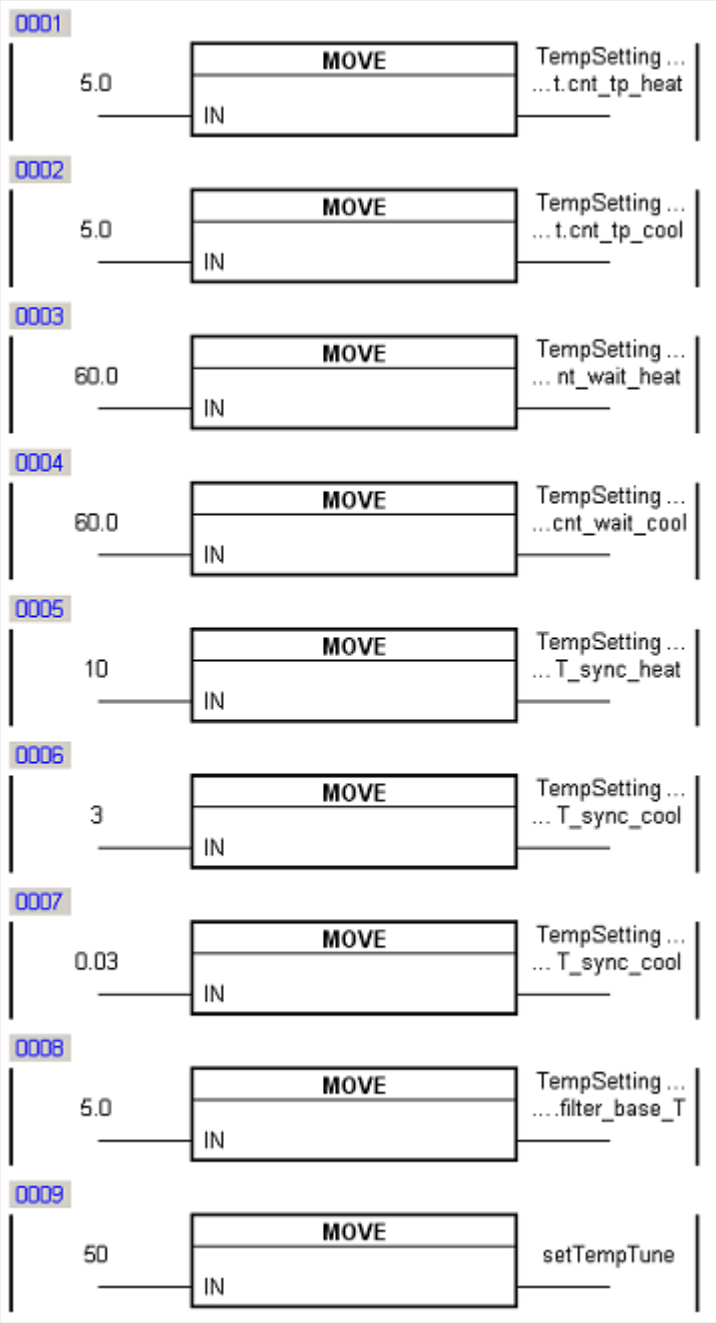


Fig. 82: B&R simulation model - Initialization routine1

Ladder diagram: Initialization routine

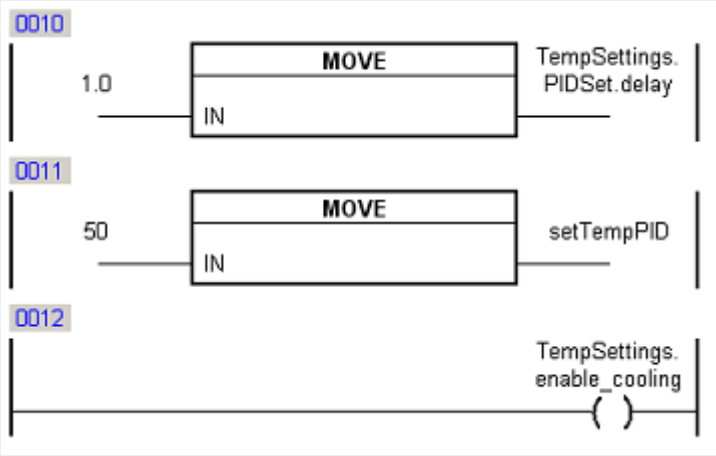


Fig. 83: B&R simulation model - Initialization routine2

Ladder diagram:

The read-in temperature *aiActTemp* must be converted from data type INTEGER to the data type REAL. To convert the unit from 1/10 °C to 1 °C, the value is divided by 10. *actTempZone1* is then the controlled variable that is connected to the *Temp* input of the LCRTempTune() function block and LCRTempPID(). Furthermore, an additional logic operation is present to automate switching from LCRTempTune() to the LCRTempPID() after the tuning.

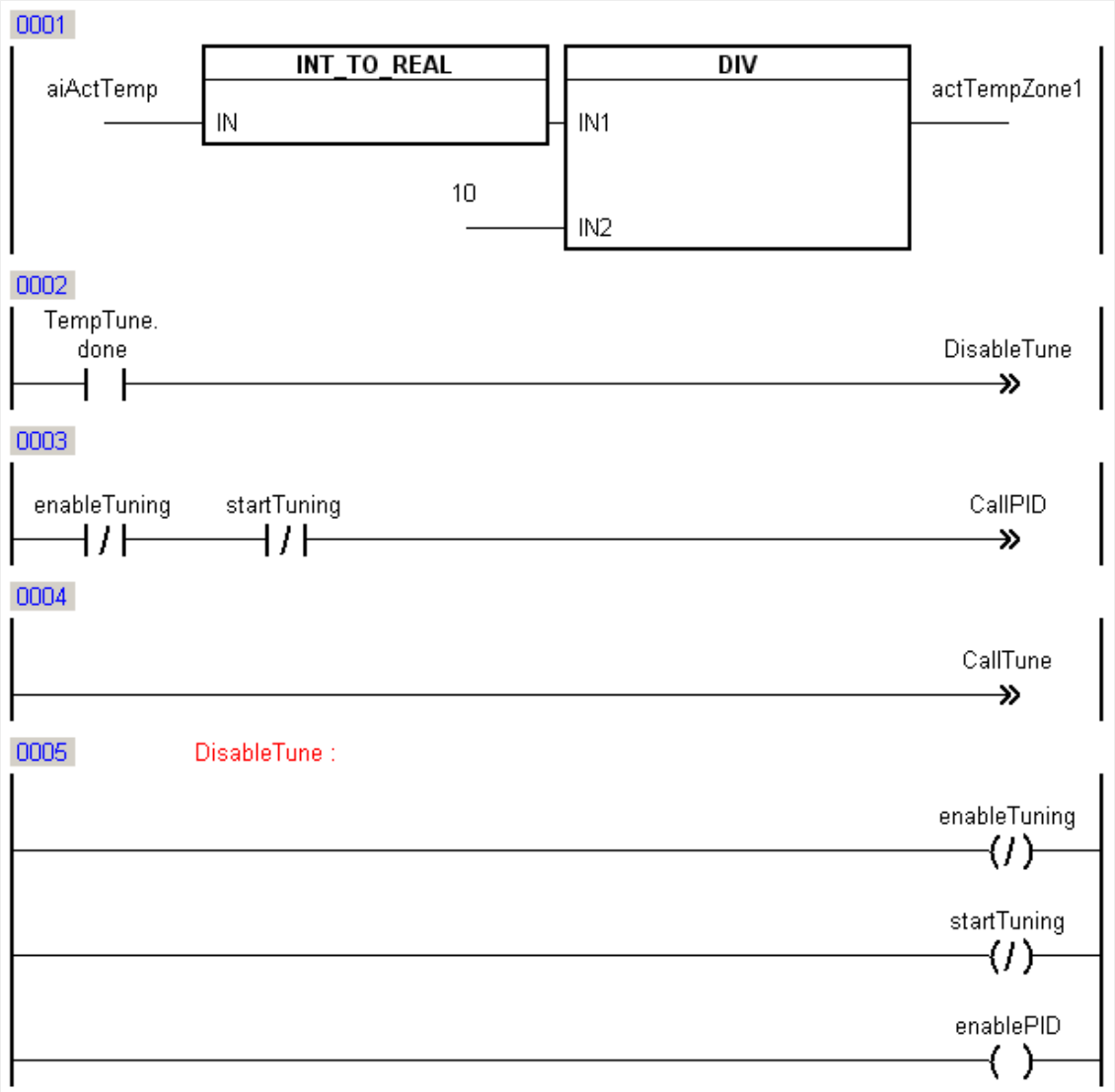


Fig. 84: B&R simulation model - Switching logic

Ladder diagram: LCRTempTune()

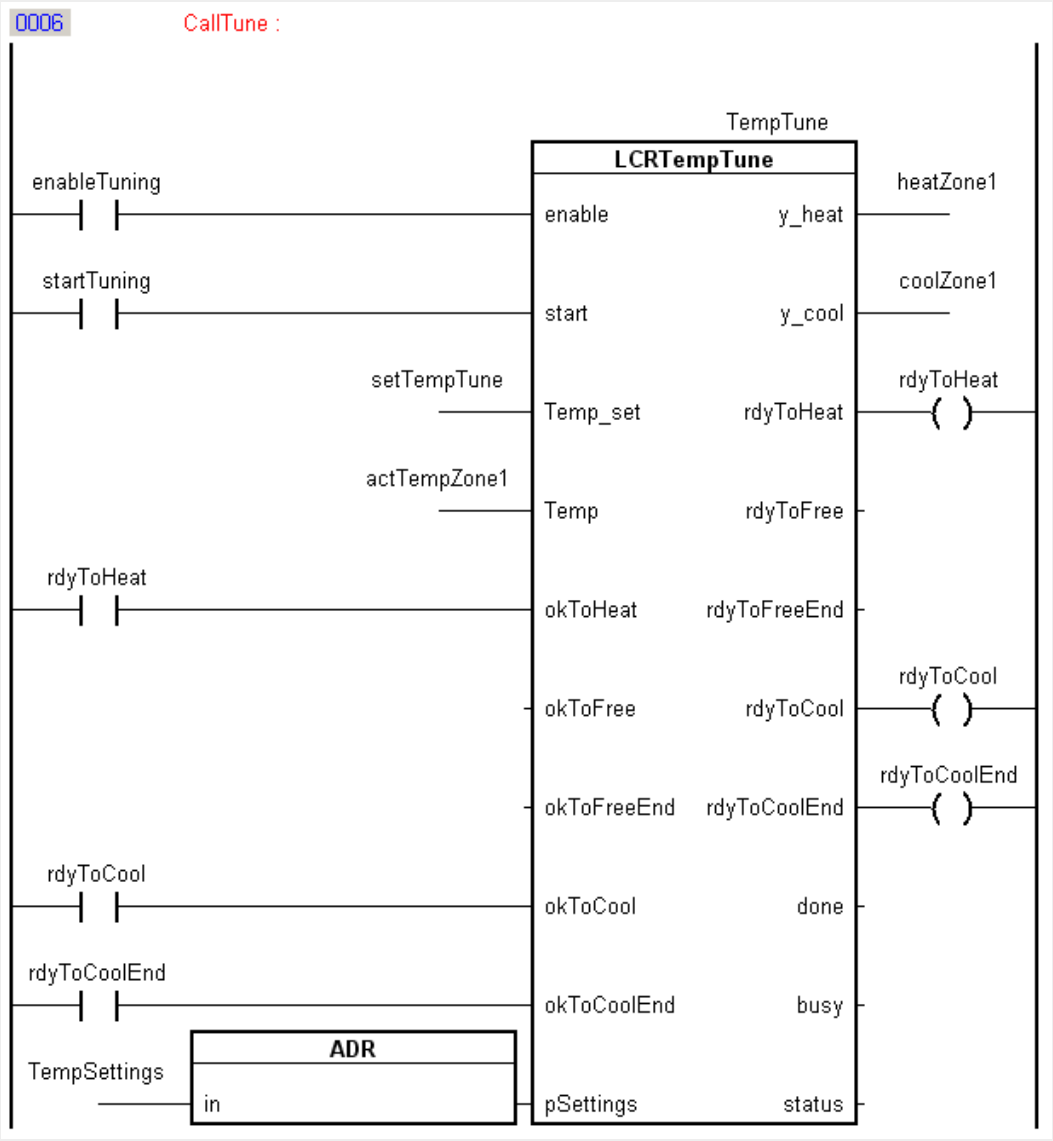


Fig. 85: B&R simulation model - LCRTempTune()

Ladder diagram: LCRTempPID()

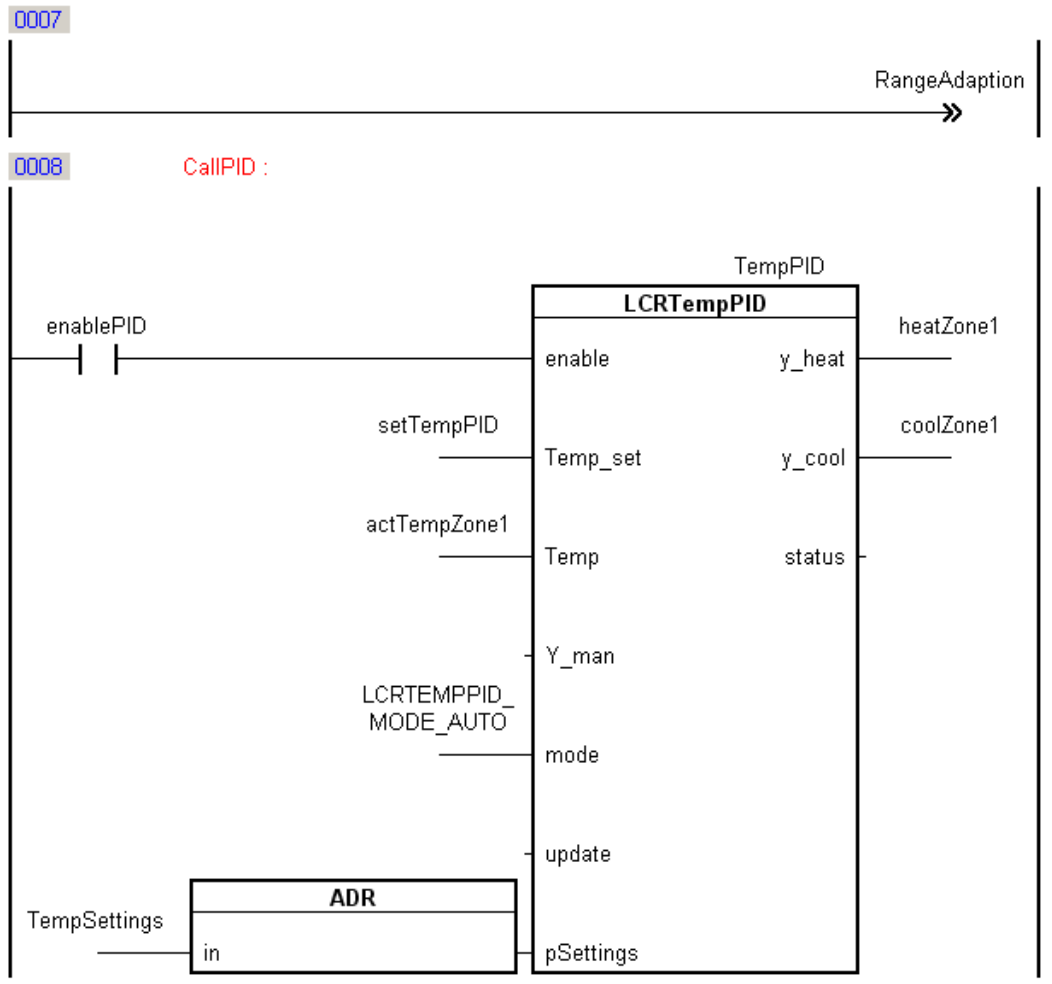


Fig. 86: B&R simulation model - LCRTempPID()

The value range of the outputs heatZone1 and coolZone1 is scaled to the value range of the analog outputs (0 - 32767) and converted from the data type REAL to INT.

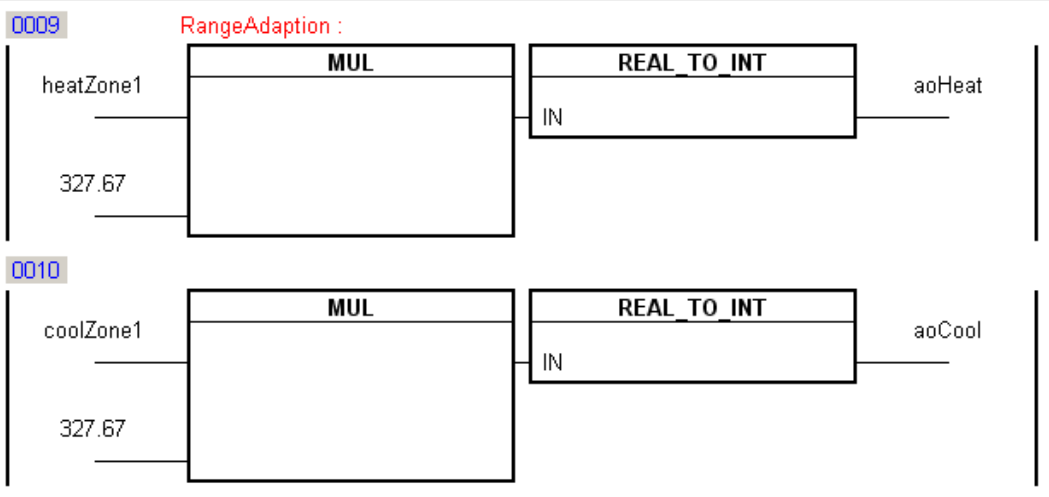


Fig. 87: B&R simulation model - Manipulated variable scaling

Variable declaration:

Name	Type	Scope	Attribute	Value
LCRTEMPPID_MODE_AUTO	UDINT	global	constant	1
TempPID	LCRTempPID	local	memory	
TempSettings	lcrtemp_set_typ	local	memory	
TempTune	LCRTempTune	local	memory	
actTempZone1	REAL	local	memory	
aiActTemp	INT	local	memory	
aoCool	INT	local	memory	
aoHeat	INT	local	memory	
coolZone1	REAL	local	memory	
enablePID	BOOL	local	memory	
enableTuning	BOOL	local	memory	
heatZone1	REAL	local	memory	
rdyToCool	BOOL	local	memory	
rdyToCoolEnd	BOOL	local	memory	
rdyToHeat	BOOL	local	memory	
setTempPID	REAL	local	memory	
setTempTune	REAL	local	memory	
startTuning	BOOL	local	memory	
zone1	LCRSimModExt	local	memory	

Fig. 88: B&R simulation model variable declaration

Trace of the tuning with subsequent activation of the controller:

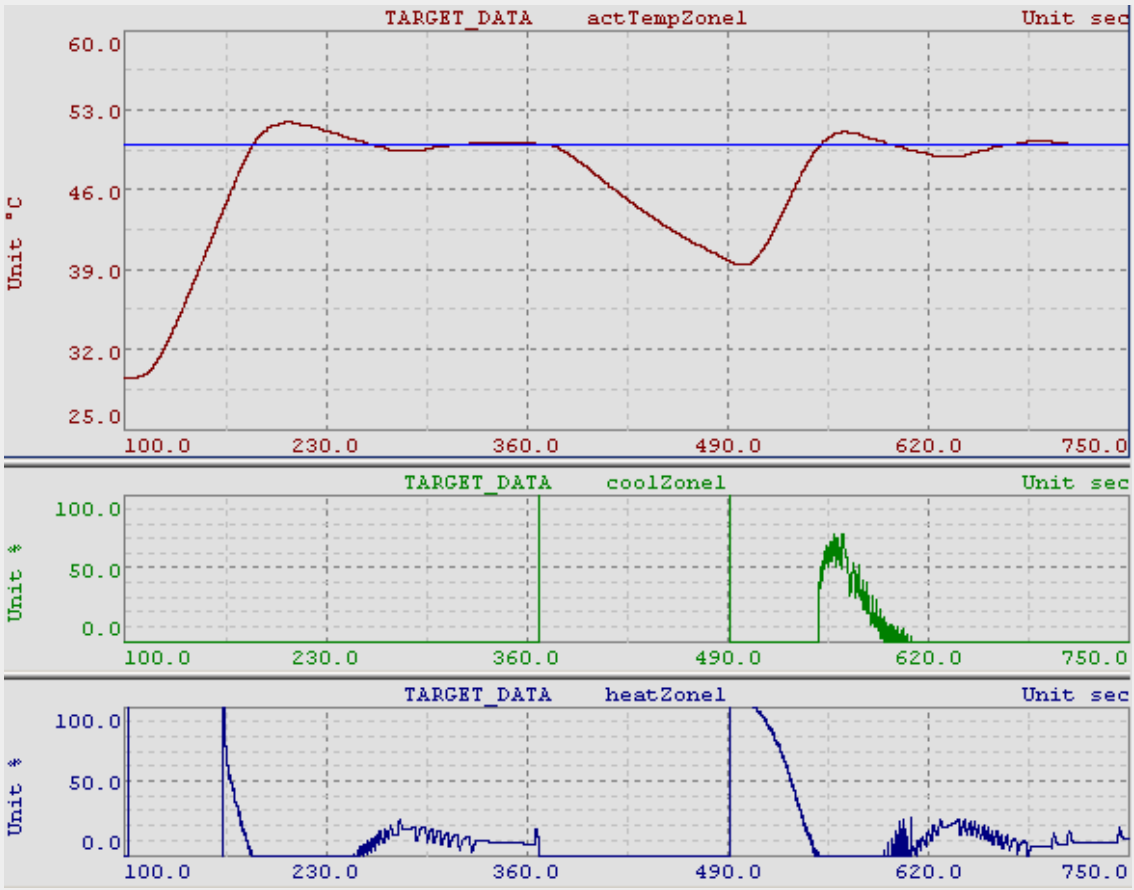


Fig. 89: Trace B&R simulation model - Tuning

PID parameters determined during auto-tuning:

Name	Type	Scope	Value
TempSettings	lcrtemp_set_typ	local	
PIDpara	lcrtemp_pid_opt_typ		
Kp_h	REAL		19.859
Tn_h	REAL		35.9002
Tv_h	REAL		7.53904
Kp_c	REAL		51.5133
Tn_c	REAL		39.8589
Tv_c	REAL		8.37038

Fig. 90: B&R simulation model - PID parameters

Pulse width modulation:

Pulse width modulation converts the manipulated variables (*heatZone1* and *coolZone1*) from analog to digital pulsed signals, whose pulse/pause behavior corresponds to the analog value.

Separate pulse width modulation must be implemented for each control action. The manipulated variables are each connected to the input *x* of the LCRPWM() function block. The *max_value* input corresponds to the maximum value that the manipulated variable can take on; *min_value* corresponds to the minimum value.

If analog outputs are used to control the heating and cooling of the B&R simulation model, then the pulsed digital signals from the pulse width modulation must be converted to a corresponding analog value. The digital value FALSE corresponds to the analog value 0; the digital value TRUE corresponds to the analog value 32767. This simple instruction can be implemented with the SEL() function.

Ladder diagram: Pulse width modulation

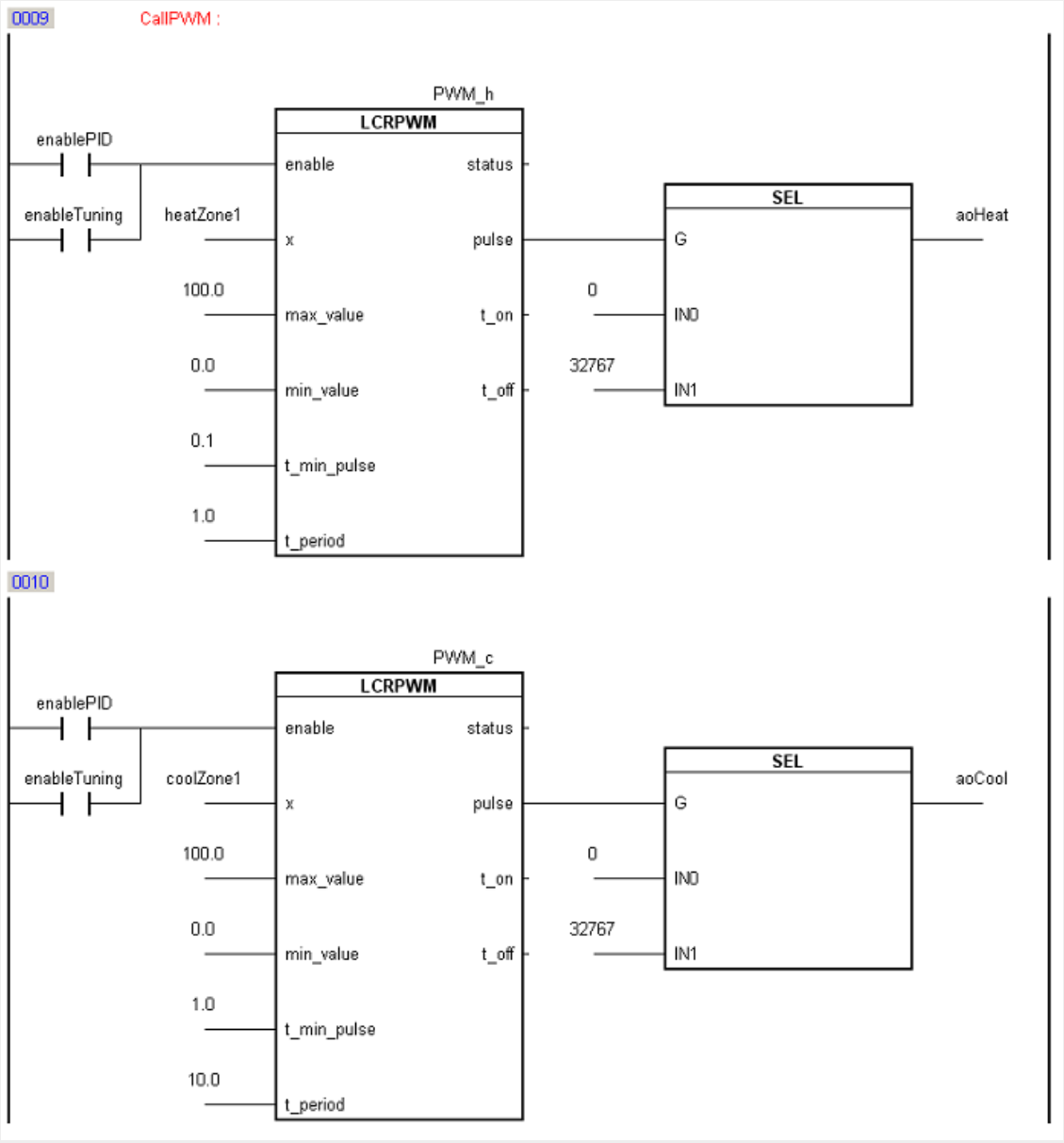


Fig. 91: B&R simulation model - Pulse width modulation

The following trace shows the heating procedure for the controlled system from 45 °C to 50 °C with a pulse width modulated control action.

- Set and actual temperatures *setTemp* and *actTemp*.
- Manipulated variable of the controller *heatZone1*.
- Output of the pulse width modulation *aoHeat*.

The pulse/pause behavior of the *aoHeat* output corresponds to the analog manipulated variable of the controller, *heatZone1*.

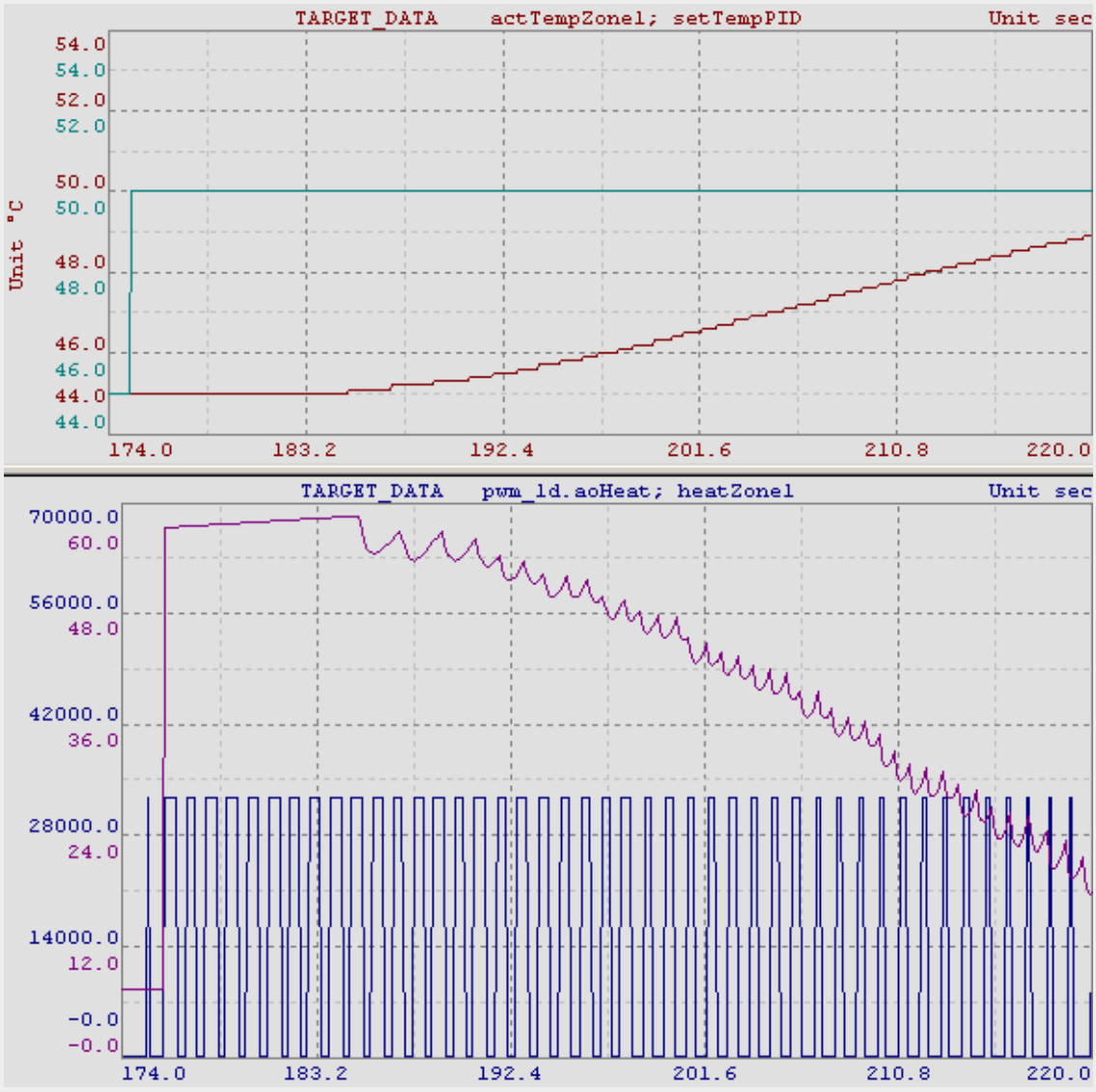


Fig. 92: Trace B&R simulation model - Pulse width modulation

Overview of training modules

TM200 – B&R Company Presentation **
TM201 – B&R Product Spectrum **
TM210 – The Basics of Automation Studio
TM211 – Automation Studio Online Communication
TM212 – Automation Target **
TM213 – Automation Runtime
TM220 – The Service Technician on the Job
TM223 – Automation Studio Diagnostics
TM230 – Structured Software Generation
TM240 – Ladder Diagram (LAD)
TM241 – Function Block Diagram (FBD)
TM246 – Structured Text (ST)
TM247 – Automation Basic (AB)
TM248 – ANSI C
TM250 – Memory Management and Data Storage
TM260 – Automation Studio Libraries I
TM261 – Closed Loop Control with LOOPCONR

TM400 – The Basics of Motion Control
TM410 – The Basics of ASiM
TM440 – ASiM Basic Functions
TM441 – ASiM Multi-Axis Functions
TM445 – ACOPOS ACP10 Software
TM450 – ACOPOS Control Concept and Adjustment
TM460 – Starting up Motors

TM500 – The Basics of Integrated Safety Technology
TM510 – ASiST SafeDESIGNER

TM600 – The Basics of Visualization
TM610 – The Basics of ASiV
TM630 – Visualization Programming Guide
TM640 – ASiV Alarm System
TM650 – ASiV Internationalization
TM660 – ASiV Remote
TM670 – ASiV Advanced

TM700 – Automation Net PVI
TM710 – PVI Communication
TM711 – PVI DLL Programming
TM712 – PVIServices
TM730 – PVI OPC

TM800 – APROL System Concept
TM810 – APROL Setup, Configuration and Recovery
TM811 – APROL Runtime System
TM812 – APROL Operator Management
TM813 – APROL XML Queries and Audit Trail
TM830 – APROL Project Engineering
TM840 – APROL Parameter Management and Recipes
TM850 – APROL Controller Configuration and INA
TM860 – APROL Library Engineering
TM865 – APROL Library Guide Book
TM870 – APROL Python Programming
TM890 – The Basics of LINUX

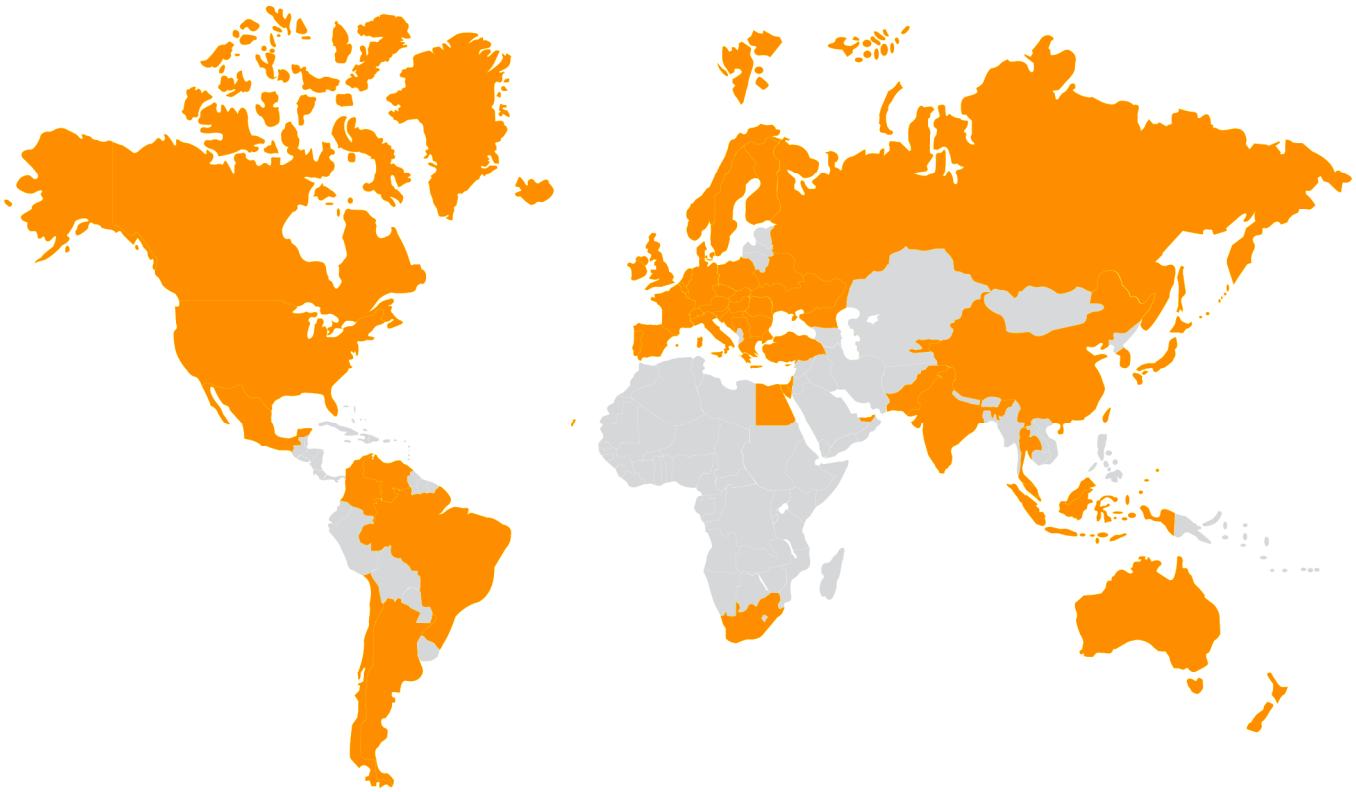
**) see Product Catalog

CORPORATE HEADQUARTERS

Bernecker + Rainer Industrie-Elektronik Ges.m.b.H.
B&R Strasse 1
5142 Eggelsberg
Austria
Tel.: +43 (0) 77 48 / 65 86 - 0
Fax: +43 (0) 77 48 / 65 86 - 26
info@br-automation.com
www.br-automation.com

TM261TRE.00-ENG 0508

140 offices in more than 55 countries - **www.br-automation.com/contact**



Australia • Argentina • Austria • Belarus • Belgium • Brazil • Bulgaria • Canada • Chile • China • Colombia • Croatia • Cyprus
Czech Republic • Denmark • Egypt • Emirates • Finland • France • Germany • Greece • Hungary • India • Indonesia
Ireland • Israel • Italy • Japan • Korea • Luxemburg • Kyrgyzstan • Malaysia • Mexico • The Netherlands • New Zealand
Norway • Pakistan • Poland • Portugal • Romania • Russia • Serbia • Singapore • Slovakia • Slovenia • South Africa
Spain • Sweden • Switzerland • Taiwan • Thailand • Turkey • Ukraine • United Kingdom • USA • Venezuela • Vietnam