Intelligent Control

Ildikó Jancskárné Anweiler

Intelligent Control

Pécs

2019

The Intelligent Control course material was developed under the project EFOP 3.4.3-16-2016-00005 "Innovative university in a modern city: open-minded, value-driven and inclusive approach in a 21st century higher education model".

EFOP 3.4.3-16-2016-00005 "Innovative university in a modern city: open-minded, value-driven and inclusive approach in a 21st century higher education model".

Ildikó Jancskárné Anweiler

Intelligent Control

Pécs

2019

Az Intelligent Control tananyag az EFOP-3.4.3-16-2016-00005 azonosító számú, „Korszerű egyetem a modern városban: Értékközpontúság, nyitottság és befogadó szemlélet egy 21. századi felsőoktatási modellben” című projekt keretében valósul meg.

EFOP-3.4.3-16-2016-00005 Korszerű egyetem a modern városban:
Értékközpontúság, nyitottság és befogadó szemlélet
egy 21. századi felsőoktatási modellben

Intelligent Control Systems

Computer Science Engineering (MSc)
University of Pécs, Faculty of Engineering and Information Technologies
Lecturer: Jancskárné Dr. Anweiler, Ildikó

Intelligent control systems

INTRODUCTIO N TO CONTROL ENGINEERING

Prerequisite lectures

Systemtheory

- Transfer characteristics of linear time-invariant systems in time and frequency domains
- Stabilty definitions

Measurement techniques

- Signal sources

Definitions

Control

•Device or mechanism installed or instituted to guide or regulate the activities or operation of an apparatus, machine, or system.

Control system

•A control system is an interconnection of components forming a system configuration that will provide a desired system response.

Plant

•A plant may be equipment, a piece of equipment, a set of machine parts functioning together, the purpose of which is to perform a particular operation. We shall call any physical object to be controlled (such as a mechanical device, a heating furnace, a chemical reactor, or a spacecraft) a plant.

Process

•A process is any operation to be controlled.

The input/output framework

•The input/output framework is used in many engineering disciplines since it allows us to decompose a system into individual components connected through their inputs and outputs. Thus, we can take a complicated system and break it down into manageable pieces. Each of these pieces has a set of inputs and outputs and, through proper design, these components can be interconnected to form the entire system.

•The input/output view is particularly useful for the special class of linear time-invariant systems.

Definitions (continue)

Self-Regulating Process

- Self-regulating processes are processes that are inherently self-regulating. Self-regulated processes have built-in feedback characteristics that cause the process to tend towards self-regulation.

Non-Self-Regulating Process

- A non-self-regulating process is one where the process does not tend towards self-regulation. These processes have no self-regulating feedback characteristics and will tend towards being unstable if not controlled externally.

Transfer characteristics

- The transfer characteristic plots the output against input. (Static I/O characteristic.)

Transfer Functions

- Any linear system is characterized by a transfer function. A linear system also has transfer characteristics. But, if a system is not linear, the system does not have a transfer function.

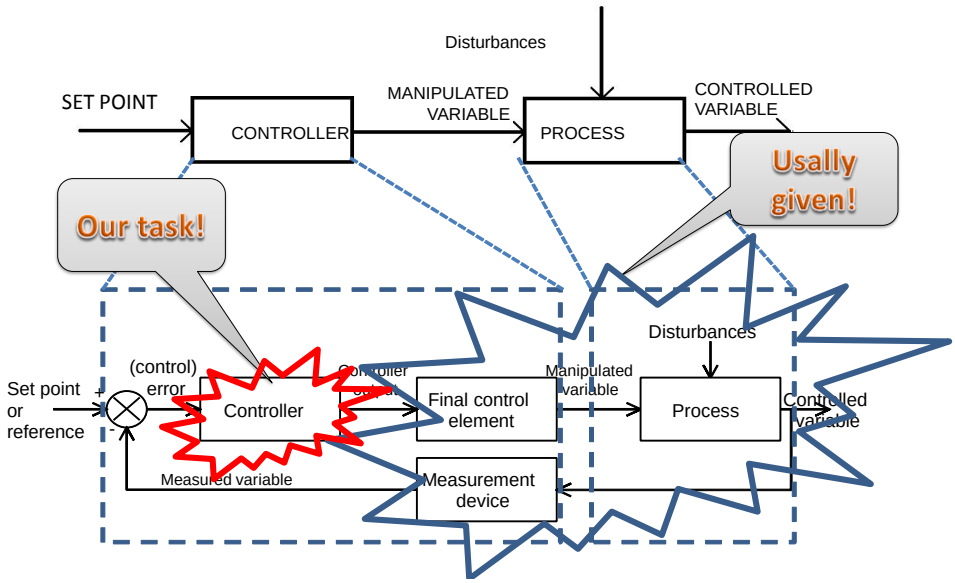
Typical test signals

- The commonly used test input signals are step functions, ramp functions, acceleration functions, pulse functions, sinusoidal functions. With these test signals, mathematical and experimental analyses of control systems can be carried out easily, since the signals are very simple functions of time.

The input/output framework

- The input/output framework is used in many engineering disciplines
- It allows us to decompose a system into individual components connected through their inputs and outputs.
- We can take a complicated system and break it down into manageable pieces.
- Each of these pieces has a set of inputs and outputs
- These components can be interconnected to form the entire system.
  The input/output view is particularly useful for the special class of linear time-invariant systems.

THE CONTROL SYSTEM AND SUBSYSTEMS



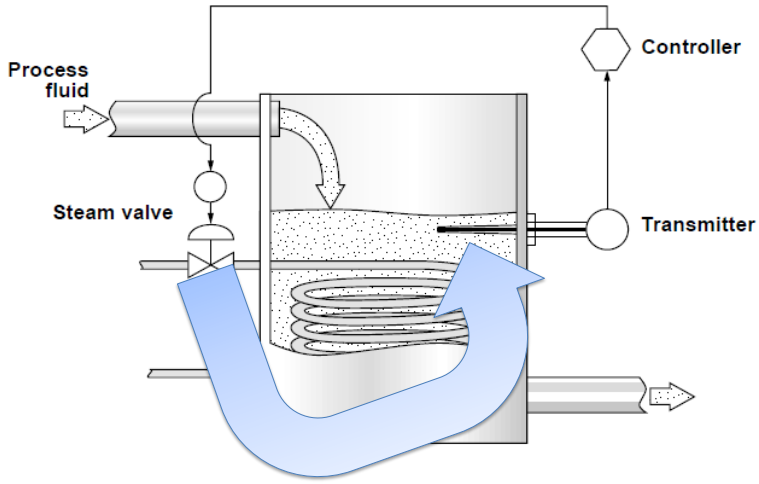
The general control problem:

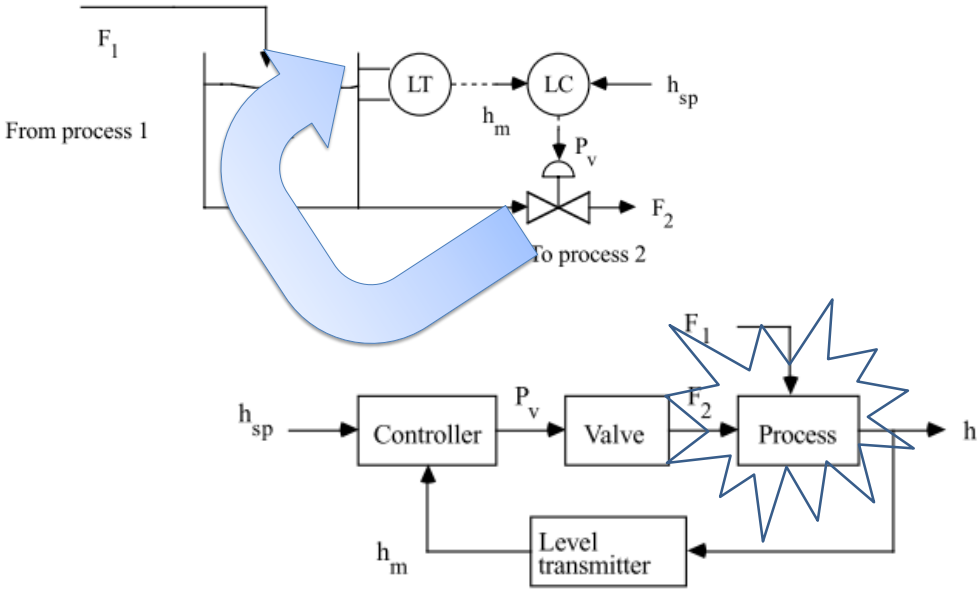
- How to construct a Controller (C), to a given Plant (P), so that the specifications on how we would like the control system to behave, be satisfied?
- Classical control problems is limited so that C and P are linear and specifications simply represent, for example, stability, rise time, and overshoot.
- Conventional control systems are designed today using mathematical models of physical systems. A mathematical model, which captures the dynamical behavior of interest is chosen and then control design techniques are applied, to design the mathematical model of an appropriate controller.
- The controller is then realized via hardware or software and it is used to control the physical system.
- The procedure may take several iterations.

The controlled process

- Process: continuously operating dynamical system
- The controller design and tuning procedure requires some information over the process behavior
- The mathematical model of the system must be "simple enough" so that it can be analyzed with available mathematical techniques, and "accurate enough" to describe the important aspects of the relevant dynamical behavior. It approximates the behavior of a plant in the neighborhood of an operating point.
- Robustness of a controller: the controller properties do not change much if applied to a system slightly different from the mathematical one used for its synthesis.
- (No real physical system truly behaves like the series of differential equations used to represent it mathematically.)

Plant or Controlled Process: all from the actuator to the sensor





The controller design and tuning procedures

PID algorithm

1. Classical control, Conventional control
   - selecting the transfer characteristic of the controller from a well kown set of algorithms (similar algorithm for various processes)
   - Setting parameters (tuning)
     - constant , gain-sceduling or adaptive

SISO models

2. Modern control, model based control, intelligent control
   - determining the control algorithm (synthesis) and setting parameters (constant or adaptive)
     - Optimal control (LQG controller) : finding a control law for a given system such that a certain optimality criterion is achieved. A control problem includes a cost functional that is a function of state and control variables.
     - MPC: Model Predictive Control: can handle multi-input multi-output (MIMO) systems that have interactions between their inputs and outputs, can handle constraints, has preview capabilities. In process industries, automotive and aerospace industries.
     - Intelligent control uses various AI computing approaches like artificial neural networks, Bayesian probability, fuzzy logic, machine learning, evolutionary computation and genetic algorithms to control a dynamic system.

LQG, MPC, Fuzzy

MIMO models

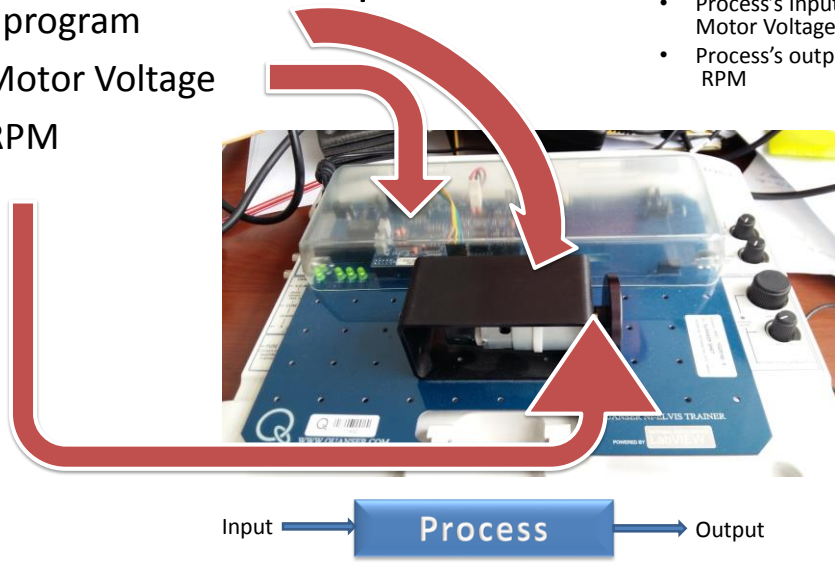
From the control point of view

- Controller output = Process input
  - Manipulated variable
  - Controlled variable

Example 1.
DC motor speed control

- Plant: Motor
- Process: DC motor speeds up
- Process's Input: Motor Voltage
- Process's output : RPM

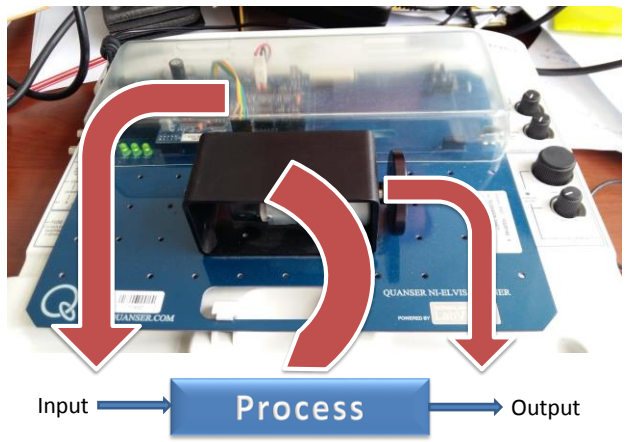
- In Control program
- Writing : Motor Voltage
- Reading: RPM

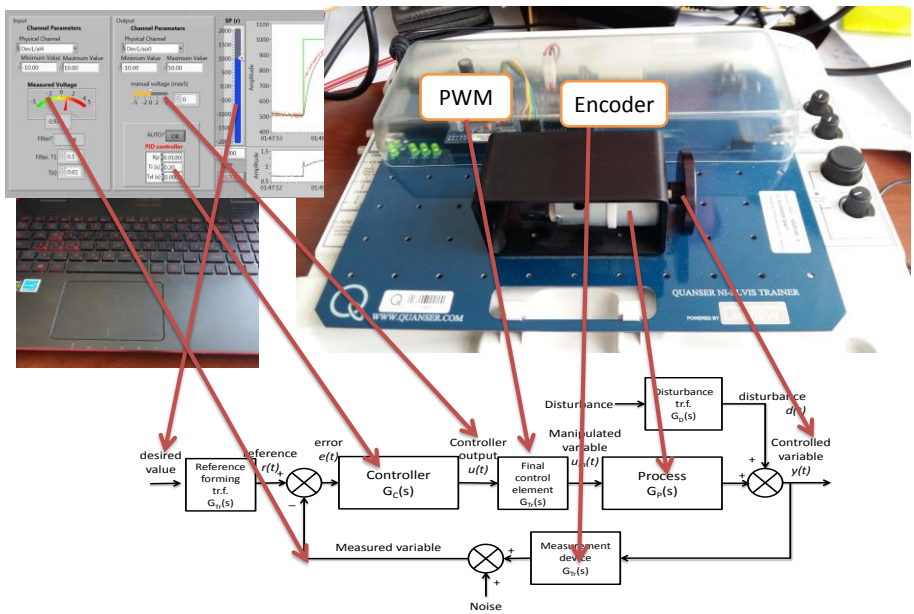


Example 1.

- Process: DC motor
- Process's Input: Motor Voltage
- Process's output : speed, RPM

- In Control program
- Writing : Motor Voltage
- Reading: RPM

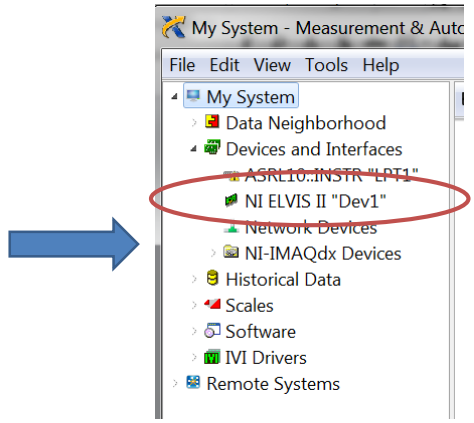




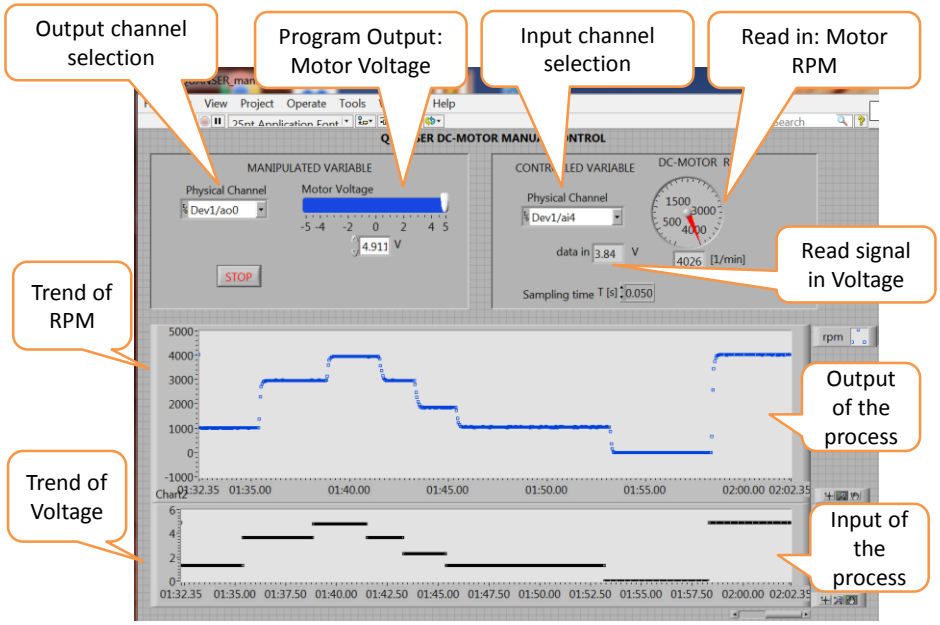
Block diagram

Test it!

- Connections
  - 1. Power connections
  - 2. NI ELVIS II to USB port
- Run NI-MAX to ensure that the ELVIS II is „Dev1”
- Run DC_QUANSER_manual.vi

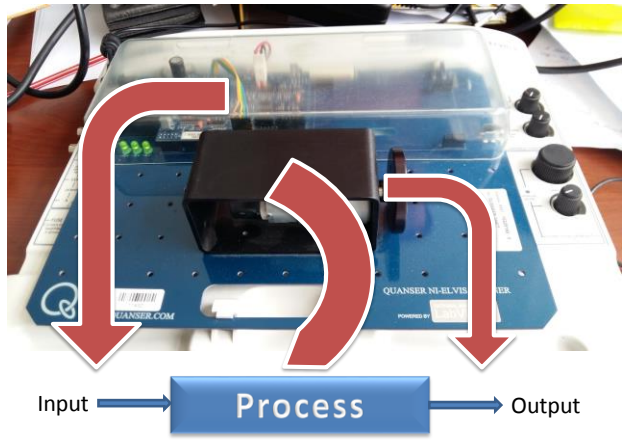


The front panel view of the control program

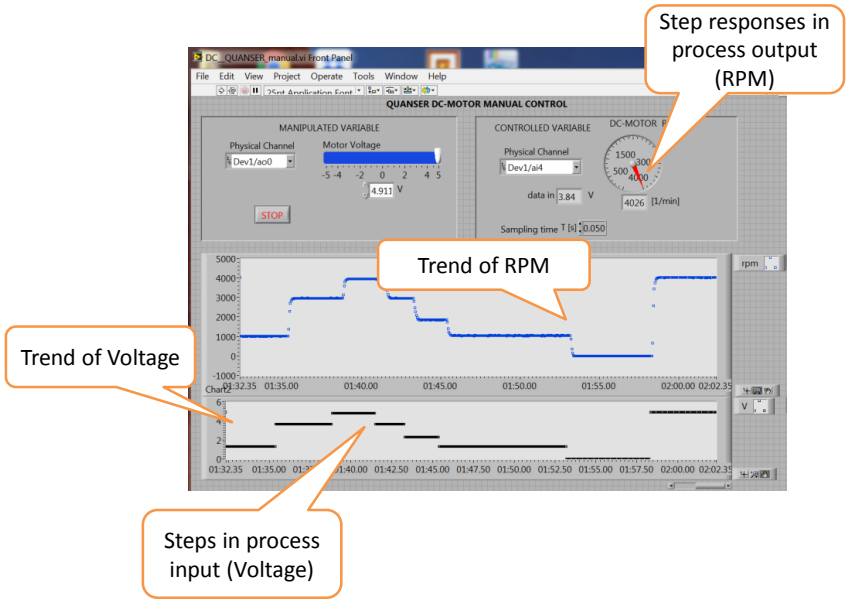


Change the motor voltage step like manually

- The readings are the step responses of the process (the speed of the motor)

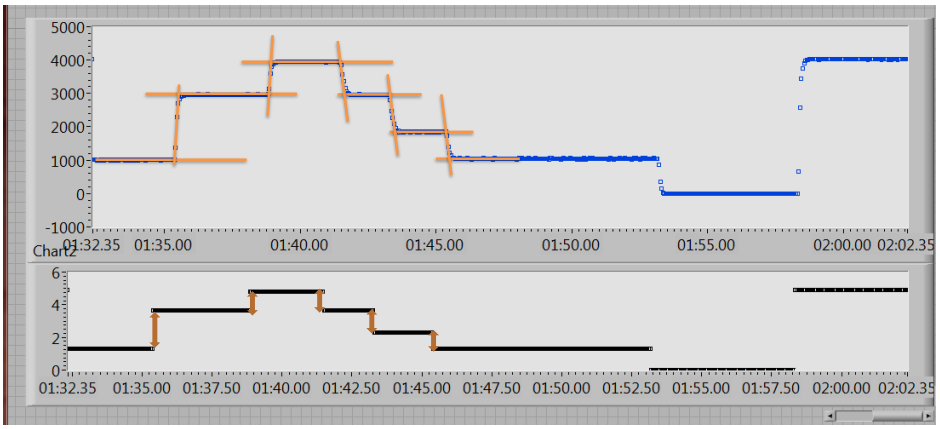


Step responses of the process



What can we read from the step responses?

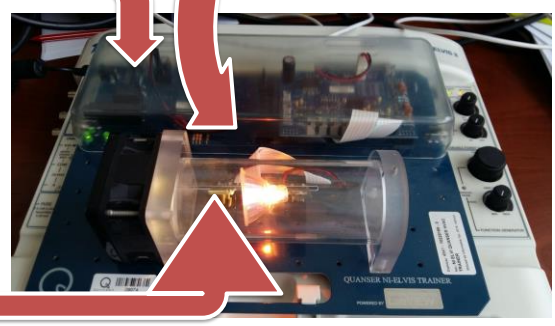
- Order of the process : 1.order (type of initial rising, no overshoot, no dead time)
- Linearity of the process: linear approxamation is valid
- Gain of the process : k=……rpm/V
- Time constant of the process: T_1=……



Example 2. HVAC

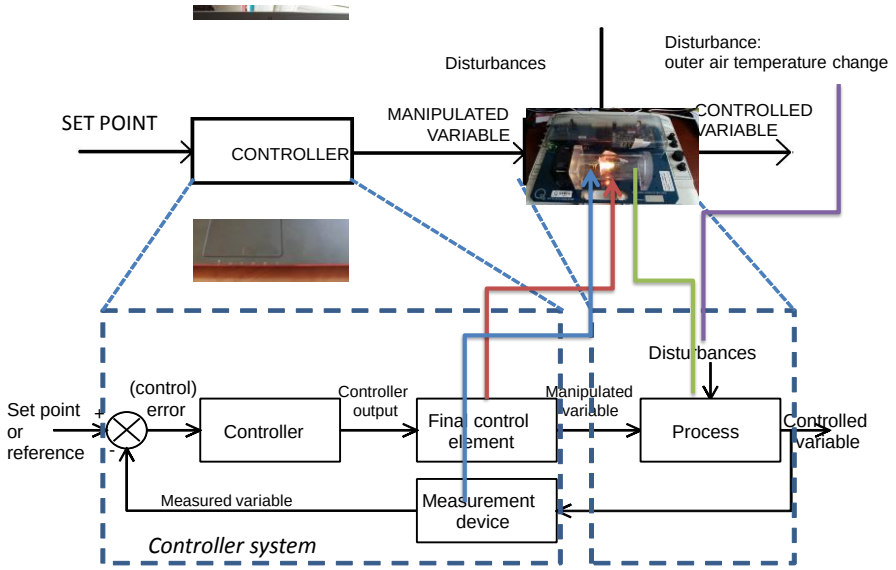
- In Control program
- Writing : Lamp's Voltage
- Reading: Temperature

- Process: HVAC modell
- Process's Input: Heating power
- Process's output : Cell's Temperature



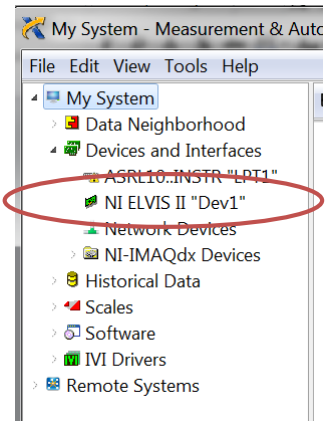
Input → Process → Output

THE CONTROL SYSTEM AND SUBSYSTEMS



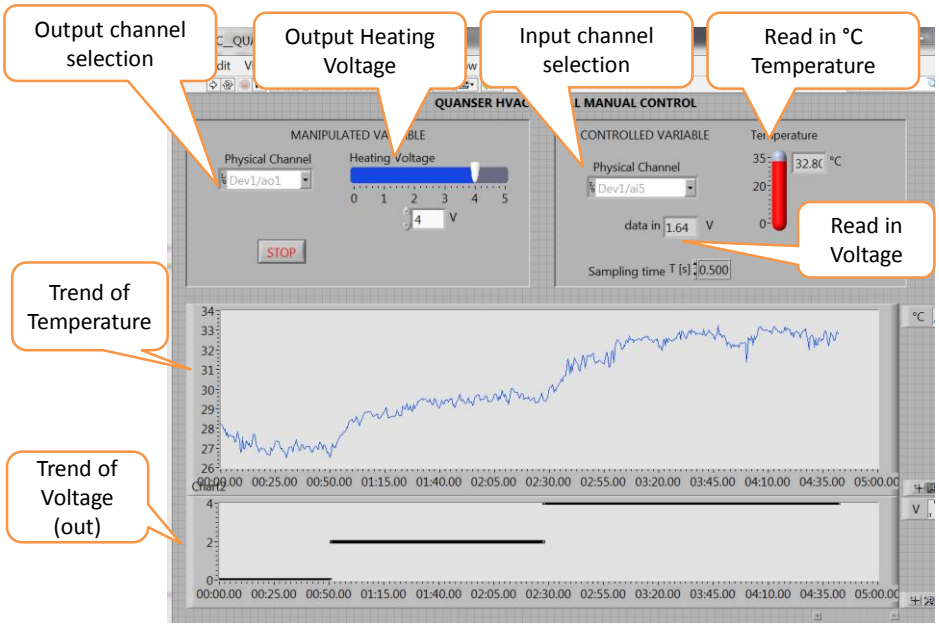
Test it!

- Connections
  - 1. Power connections
  - 2. NI ELVIS II to USB port
- Run NI-MAX to ensure that the ELVIS is Dev1

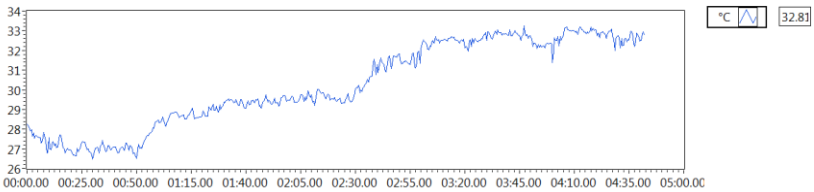


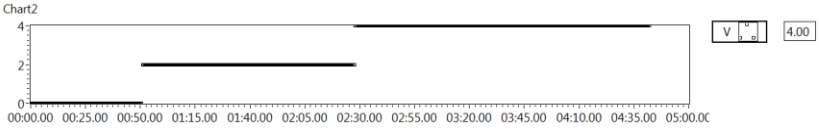
- Run HVAC_QUANSER_manual.vi

The front panel view



Noise in the temperature readings
What can we read from the step responses?

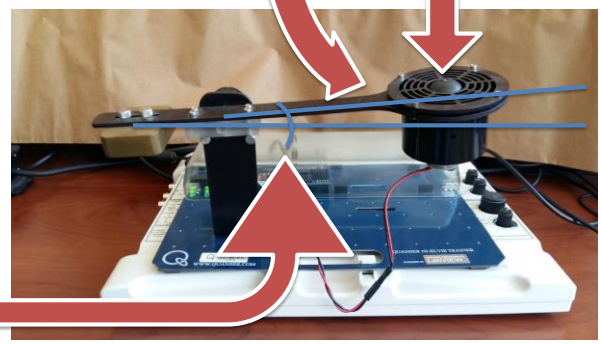




Example 3.
Vertical Take-On Modell

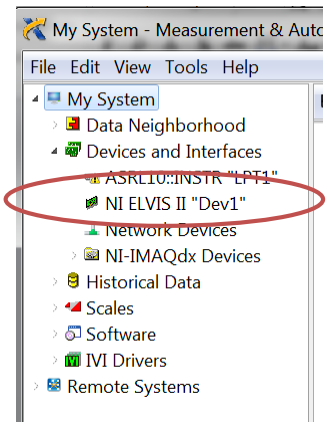
- Process: VTOL modell
- Process's Input: Motor voltage
- Process's output : Position

- In Control program
- Writing : Motor Voltage
- Reading: position



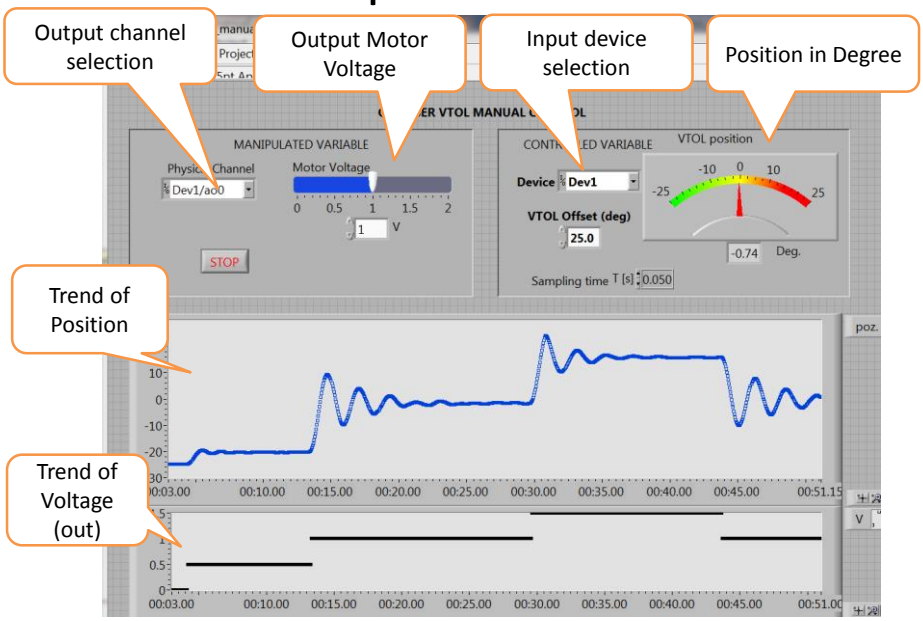
Test it!

- Connections
  - 1. Power connections
  - 2. NI ELVIS II to USB port
- Run NI-MAX to ensure that the ELVIS II is „Dev1"

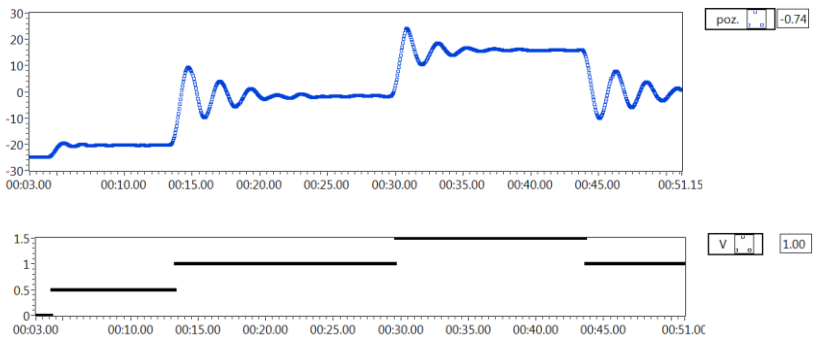


- Run VTOL_QUANSER_manual.vi

The front panel view



Step responses: damped oscillation



System Modeling

- The whole premise behind Classical Control is that the system to be controlled has to be rigidly modeled.
- Usually, this is accomplished by approximating the system to a first-order or second-order transfer functions (with dead-time).
- First-Order System: $G(s) = \frac{k}{Ts+1}$
- Second-Order System: $G(s) = \frac{k}{T^2s^2+2\xi Ts+1}$
- Dead-Time: $G(s) = \mathrm{e}^{-\tau s}$
- where: *k*: gain; *T*: Time constant, τ: Dead-time; ξ: Damping factor

Major types of control systems

- Man made control systems
- Natural control systems
- Mixed (combination) control systems

Man made control systems

- The system (technology) is created by human
- Example: electrical switch

Natural control systems

- Also called biological control
- This type of control is available in nature

Mixed (combination) control systems

- The system is controlled by nature (human) through man-made technology
- Example: driving a car

Mixed (combination) control systems example: Drone control

- The system is controlled by human through man-made technology

Intelligent control

- Intelligent control uses various AI computing approaches like artificial neural networks, Bayesian probability, fuzzy logic, machine learning, evolutionary computation and genetic algorithms to control a dynamic system.
- Control of a robot arm
- Control of autonomous vehicles (Intelligent UAV: Unmanned Aerial Vehicles UGV: Unmanned Ground Vehicles)
- Intelligent control systems in a smart home
- Etc.

Intelligent contol example

Winbot: Automatic window cleaner

Smart Home

- Smart home it is automation of the home, housework or household activity.
- Smart home allow you to make some access to your home via Web or a mobile app including cameras, doors, windows, locks, lights, appliances, thermostats and various sensors for security, comfort and energy management.
- A smart home connects all devices/appliances at home for sharing and also allows access from global Internet for monitoring and control to improve security, comfort and energy efficiency.

Types of control systems

By human involvement

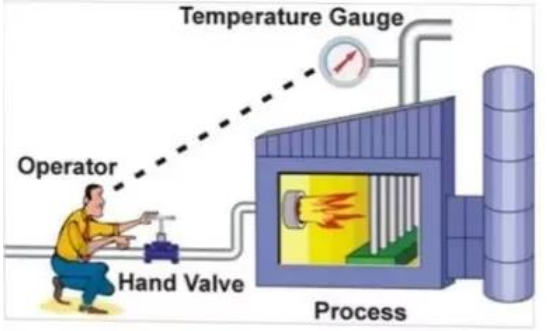
- *Manual.*
  Control operations that involve human action to make an adjustment are called manual control systems.
- *Automatic.*
  Control operations in which no human intervention is required are called automatic control systems

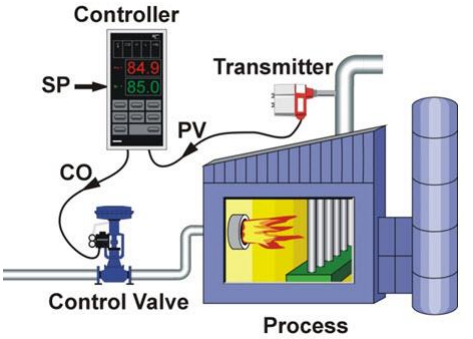
Using auxiliary energy

- *Without auxiliary energy*.
  The components of the control unit take the energy from the process to be controlled.
- *Auxiliary energy*.
  A separate power source provides the control equipment. The most common types of auxiliary energy: electric, pneumatic, hydraulic.

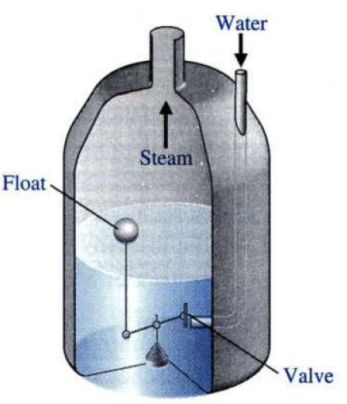
By reaction

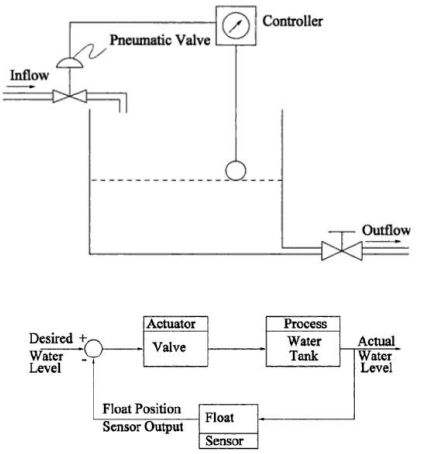
- *Open-loop control,*
  where the controlled output has no (direct) impact on the input variable.
- *Closed-loop control,*
  where the deviation of the controlled variable from the set point is used to reduce or eliminate the deviation itself.





Level control without auxilary energy



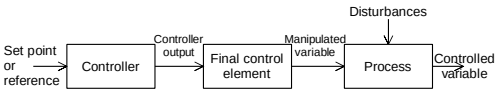


Types of control systems by reactions

Control system

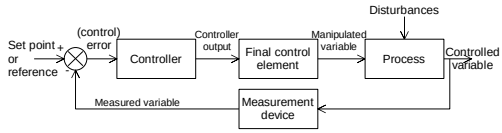
Open-loop

Systems that utilizes a device to control the process without using feedback.



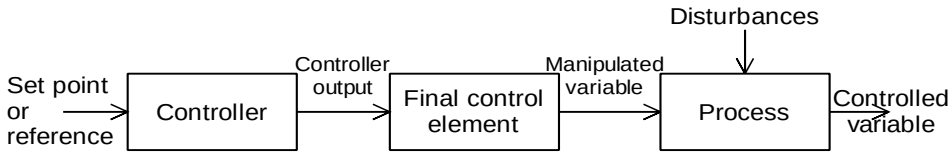
Closed-loop

Systems that uses a measurement of the output (usually a sensor) and compares it with the desired output

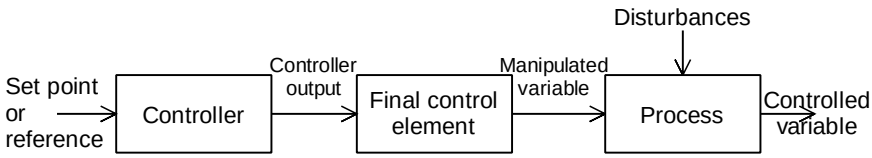


Open loop control

- an open-loop system is expected to faithfully follow its input command or set point regardless of the final result.
- Open loop control presumes existence of a model of the controlled system.
- Since the model is usually incomplete and/or inadequate, the closed loop controller is required for error compensation which uses a feedback.
- Open-loop control systems are often used with processes that require the sequencing of events with the aid of “ON-OFF” signals.



Open-loop control block diagram



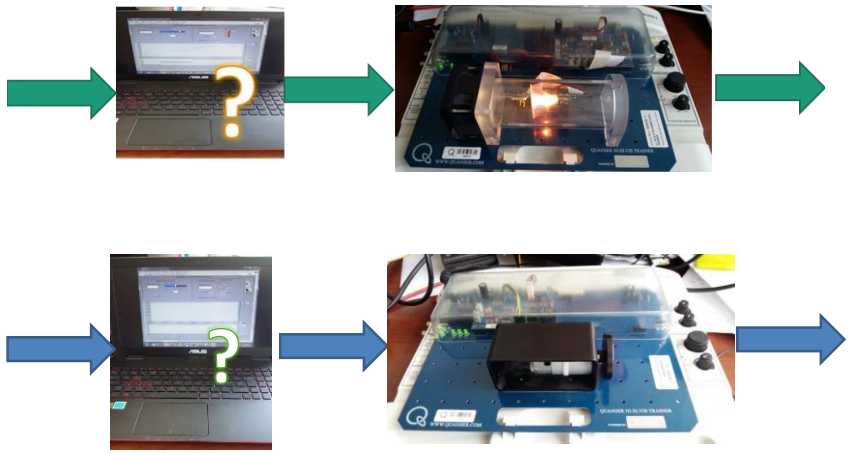
- Contains no feedback: in an open-loop control system the output is neither measured nor fed back for comparison with the input

One practical example is a common washing machine. Soaking, washing, and rinsing in the washer operate on a time basis. The machine does not measure the output signal, that is, the cleanliness of the clothes.

How can we control the speed of the DC motor model in open loop control manner?

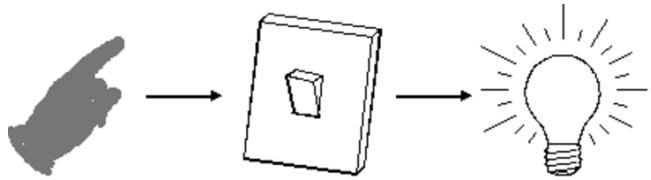
How can we control the temperature of the HVAC model in open loop control manner?

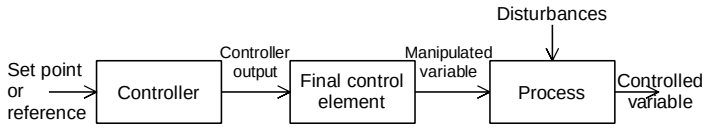
Explain!



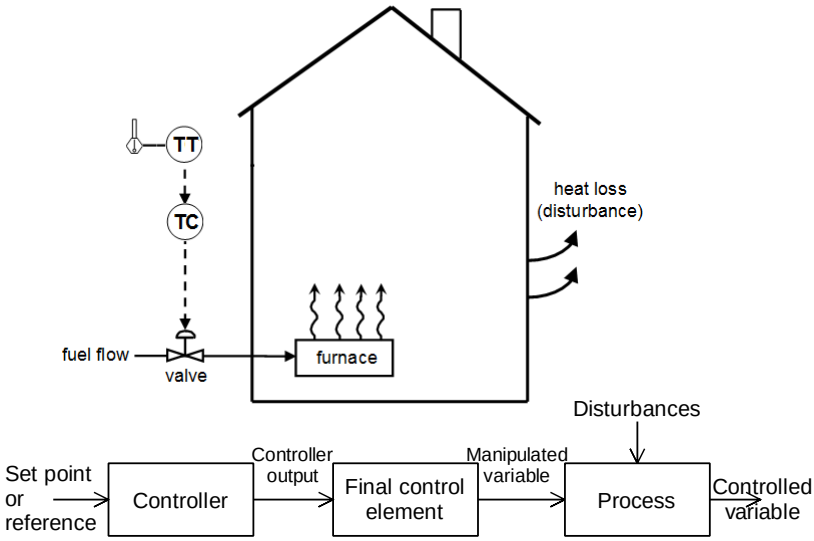
Open-loop control example

Simple electric switch





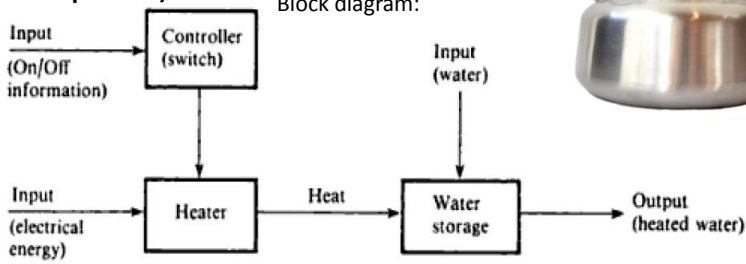
Open-loop room temperature control



Open-loop control example

- Heating water in a Kettle
  (on a hot plate)

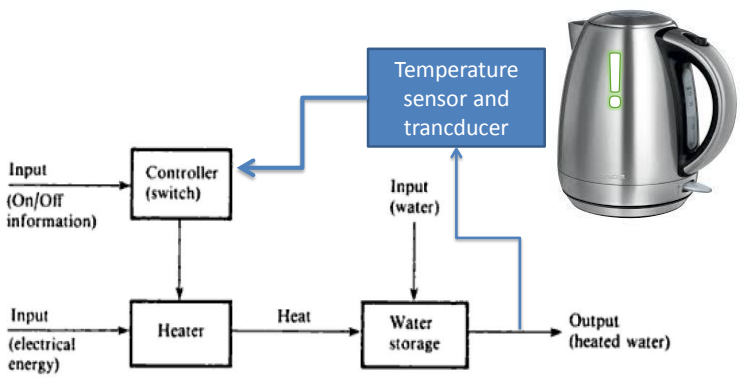
It is merely an on-off device.
Block diagram:



Source : Warwick, An Introduction to Control Systems

But, Heating water in an electric kettle ?

Feedback: closed loop control !!

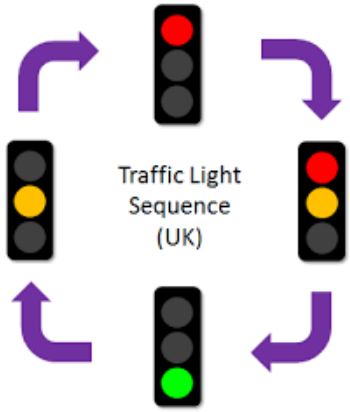


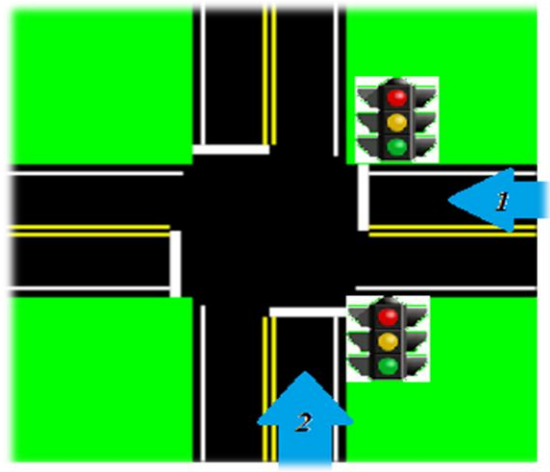
Toaster

- the **system** contains the electrical resistances of the toaster,
- the **input** is the level (time) of toasting and
- the **output** is the toasted bread.
- For the same input (time/level) the output can be more or less "toasted" depending on sever temperature, etc.
- The system is *open loop* because the input is not automatic adjusted function of the output.

Traffic light control

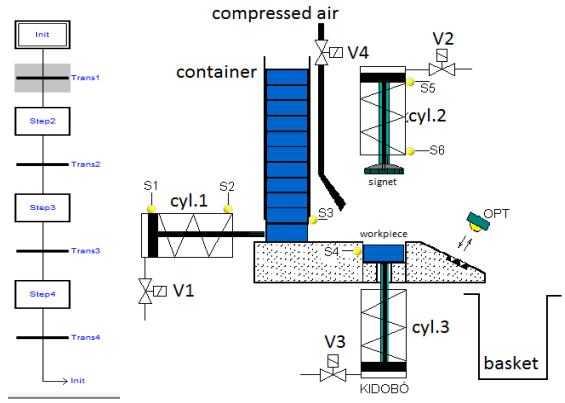
- Control type: open-loop
- Timed control



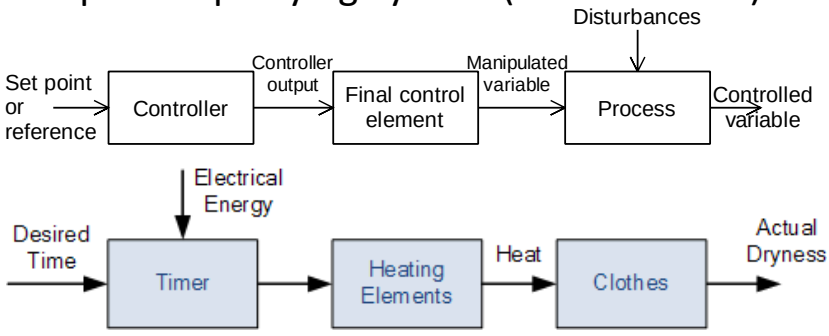


Sequential control (or batch control)

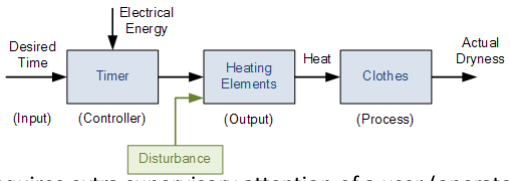
- Process bounded:
- The structure of a sequential control follows the step-by-step structure of the technology



Open-loop Drying System (timed control)



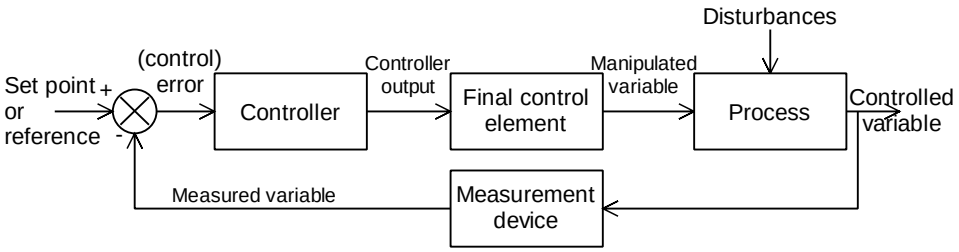
errors can disturb the drying process



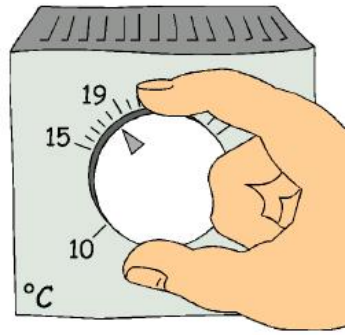
Requires extra supervisory attention of a user (operator)

Closed-loop systems

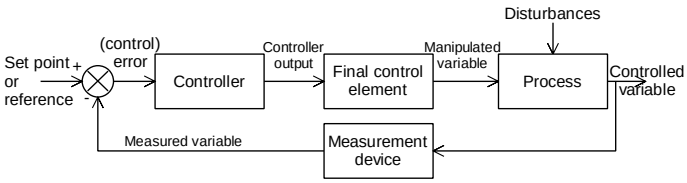
- Also known as the feedback system
- The actuating error signal, which is the difference between the input signal and the feedback signal is fed to the controller so as to reduce the error and bring the output of the system to a desired value.
- The term closed-loop control always implies the use of feedback control action in order to reduce system error.



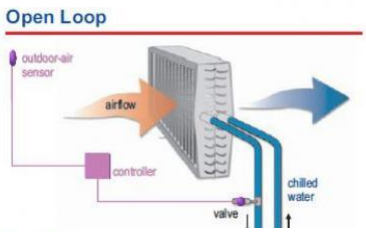
Example: air conc

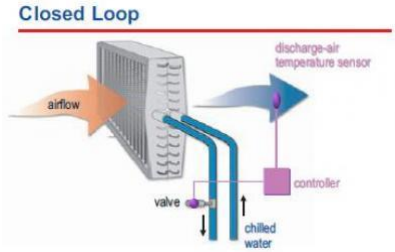


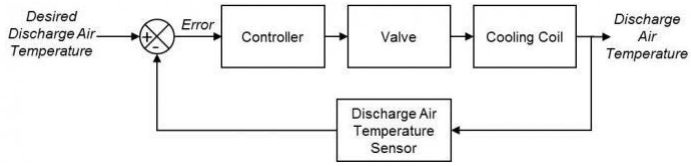
- Control type: closed-loop
- It is a self-regulating machine, performing the operation with and without the need of the human.
- This machine will keep the surrounding temperature to that of the preset value.
- Sensor is used to maintain the temperature in which the airconditioner is placed.



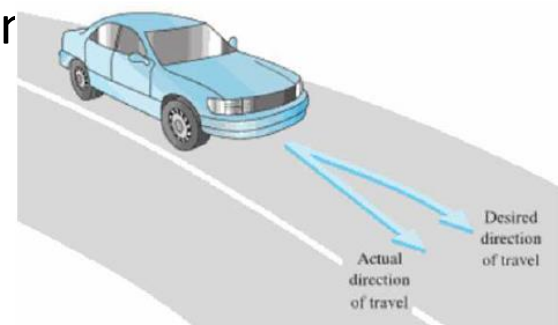
Examples: Open-Loop and Closed-Loop Air Conditioning Control Systems



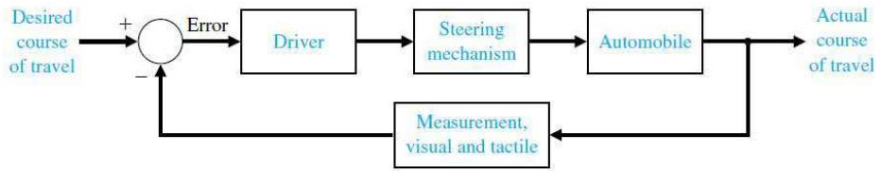




Example: drivir

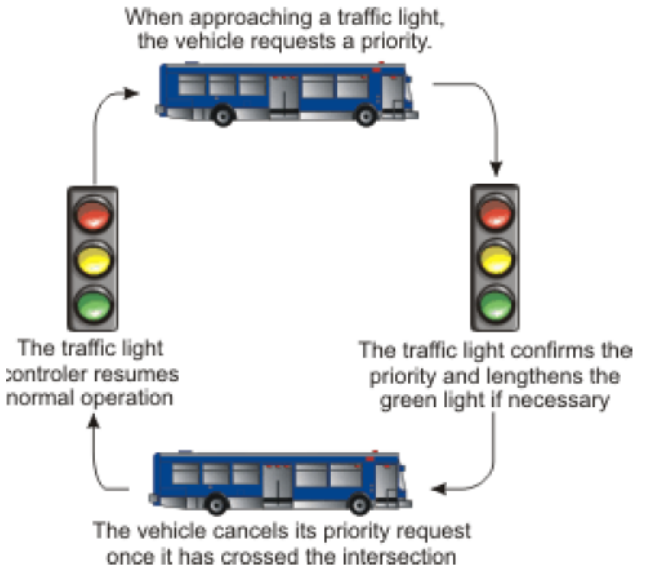


- A person steering an automobile, assuming his or her eyes are wide open, by looking at the auto's location on the road and making the appropriate adjustments
- Block diagram



Closed loop control of Traffic light

- The idea is to minimize the waiting time
- Furthermore, it is also intended to make the traffic flow smooth
- Many control techniques can be used: intelligent control system is one of them



Piping and instrumentation drawings (P&ID)

- The Instrumentation, Systems, and Automation Society (ISA) is one of the leading process control trade and standards organizations.
- The ISA has developed a set of symbols for use in engineering drawings and designs of control loops (ISA S5.1 instrumentation symbol specification).
- P&ID uses symbols and circles to indicate the components of control and lines to information links
- Readings: Process Control Fundamentals P ID.pdf & Standard Isa - Instrumentation Symbols And Identification.pdf

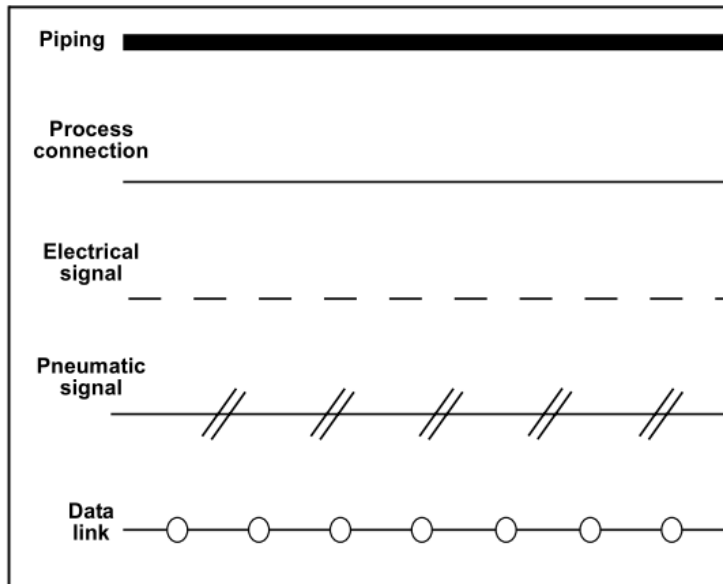
SYMBOLS

General instrument or function symbols			
	Primary location accessible to operator	Field mounted	Auxiliary location accessible to operator
Discrete instruments	1	2	3
Shared display, shared control	4	5	6
Computer function	7	8	9
Programmable logic control	10	11	12

1. Symbol size may vary according to the user's needs and the type of document.
2. Abbreviations of the user's choice may be used when necessary to specify location.
3. Inaccessible (behind the panel) devices may be depicted using the same symbol but with a dashed horizontal bar.

Source: Control Engineering with data from ISA S5.1 standard

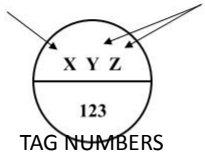
Piping and Connections



IDENTIFICATION LETTERS

*The first letter is used to designate the **measured variable***

*The succeeding letter(s) are used to designate the **function** of the component, or to **modify** the meaning of the first letter.*



TAG NUMBERS

associated with a particular control loop

Pressure

Level

Flow

Temperature

Indicator

Recorder

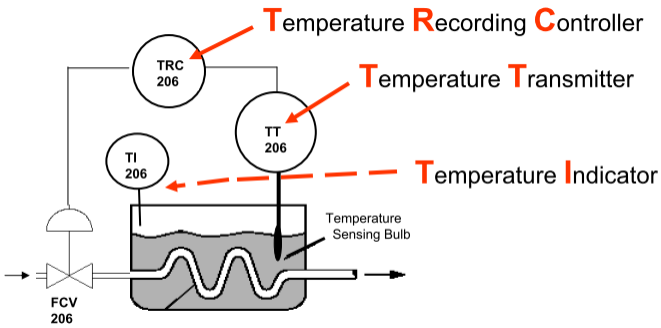
Controller

Transmitter

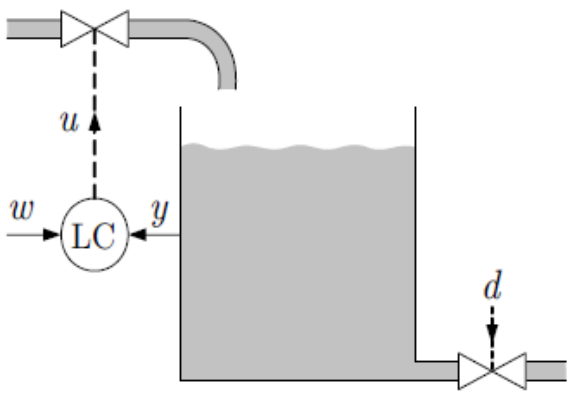
IDENTIFICATION LETTERS

	First-letter		Succeeding- Letters		
	Measured or Initiating variable	Modifier	Readout function	Output function	Modifier
A	Analysis				
C				Control	
D		Differential			
F	Flow Rate	Ratio			
H	Hand				High
I	Current		Indicate		
L	Level				Low
P	Pressure, vacuum				
Q	Quantity	Totalizer			
S		Safety		Switch	
T	Temperature			Transmit	
V	Vibration			Valve, Damper	
z	Position			Actuator	

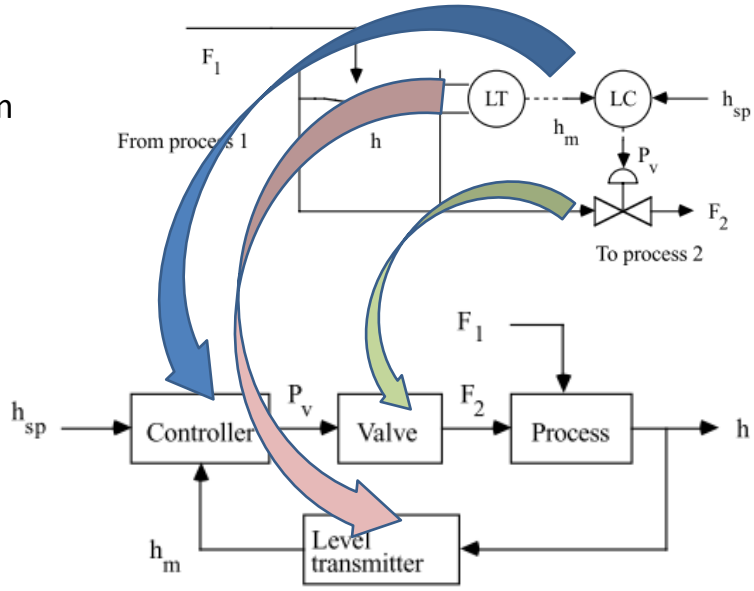
IDENTIFICATION LETTERS



Example P&ID: Level control



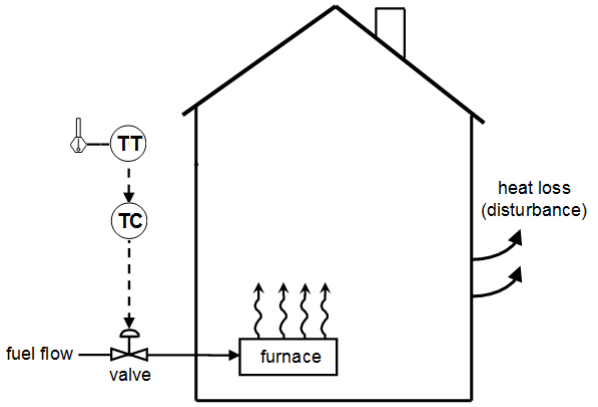
P&ID and
Block Diagram



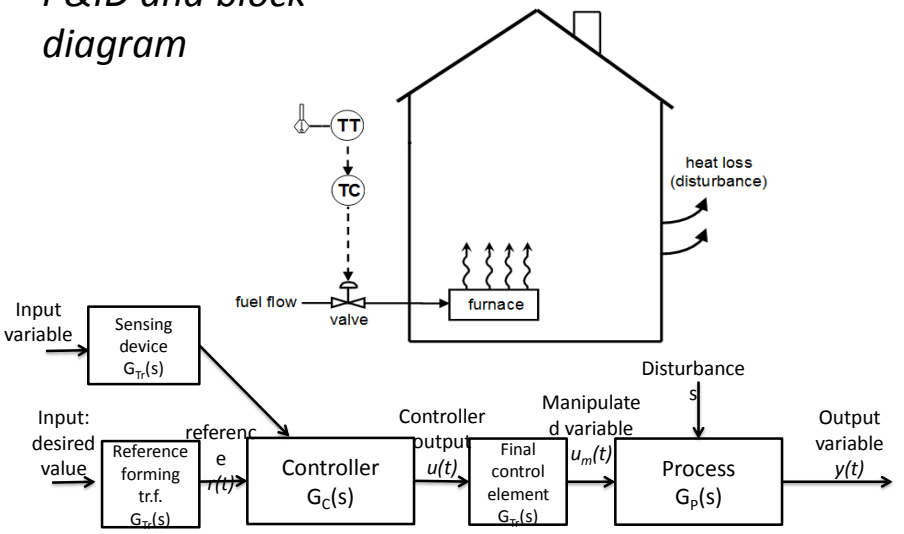
Intelligent control systems

ELEMENTS OF THE CONTROL LOOP

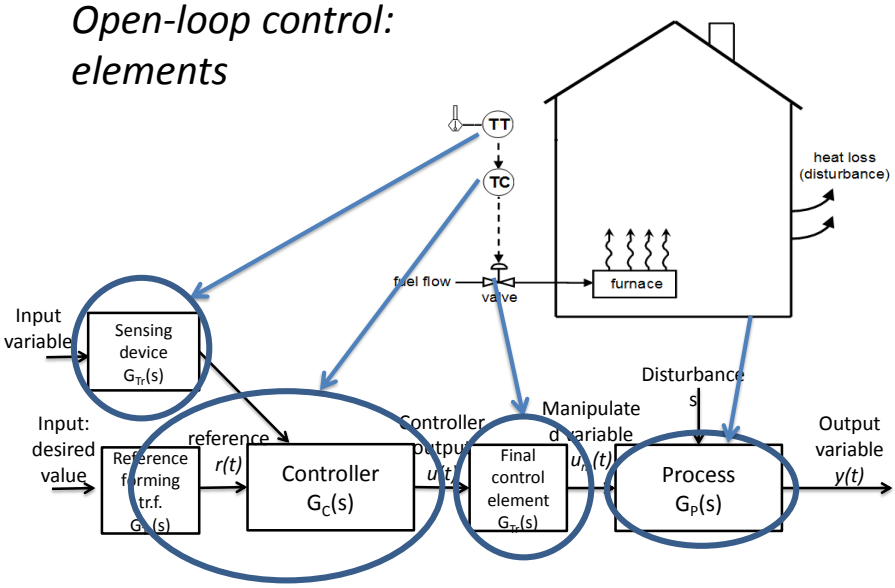
Open- or closed-loop control?
Example: home heating system (ver.1)



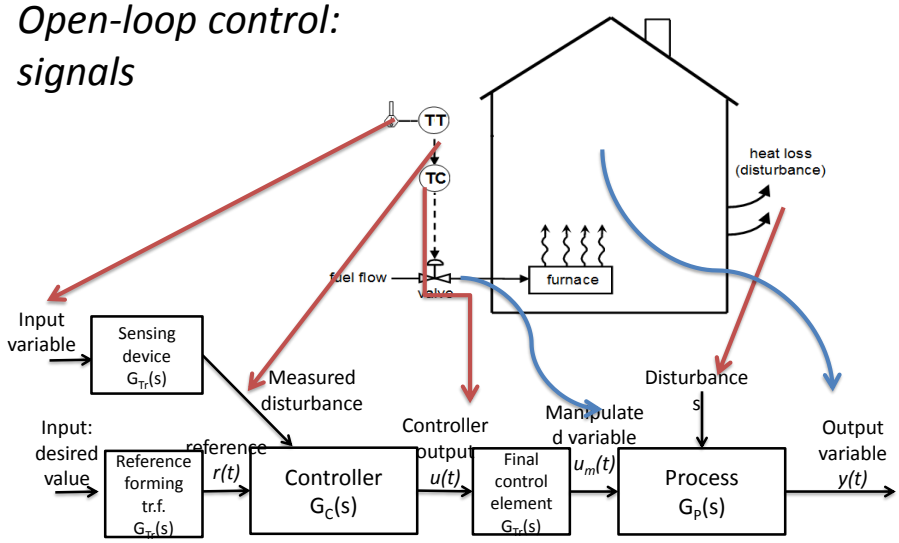
Open-loop control: P&ID and block diagram



Open-loop control: elements



Open-loop control: signals

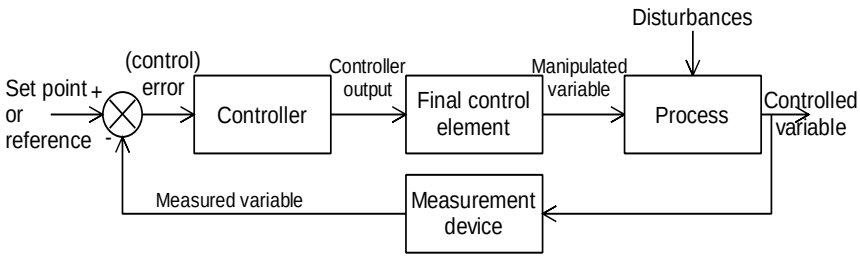


Closed-looop control

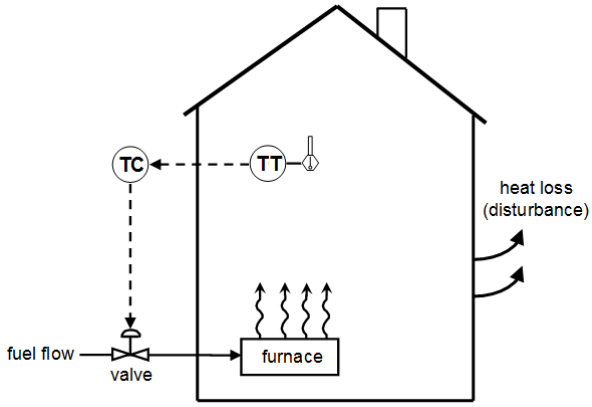
In a closed-loop control system the error signal, which is the difference between the input reference signal and the feedback signal is fed to the controller so as to reduce the error and bring the output of the system to a desired value.

Feedback makes the system response relatively insensitive to external disturbances and internal variations in system parameters.

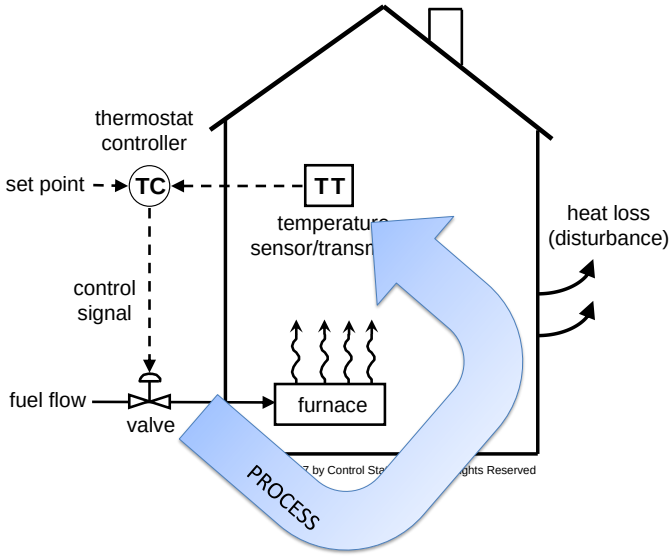
General block representation of a closed-loop control system



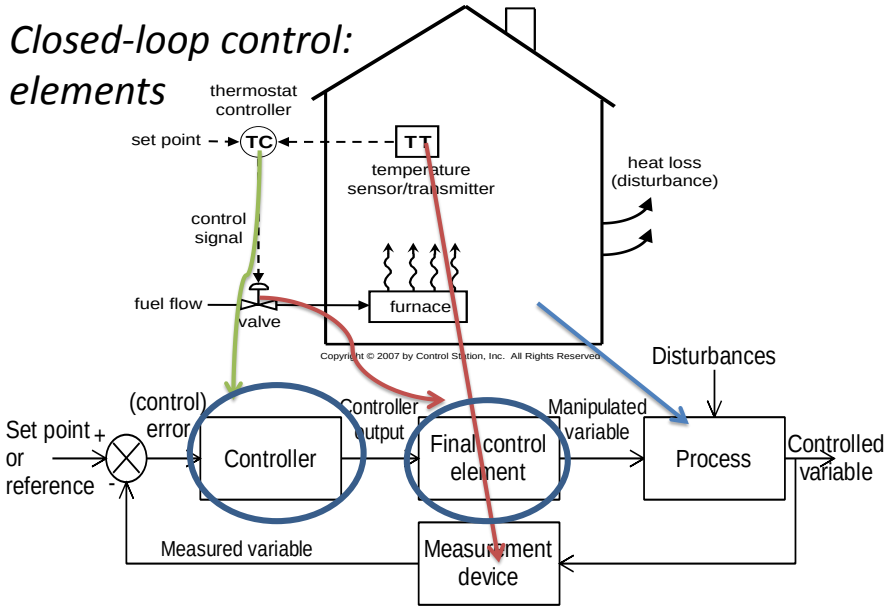
Closed-looop control example: home heating system (ver.2)



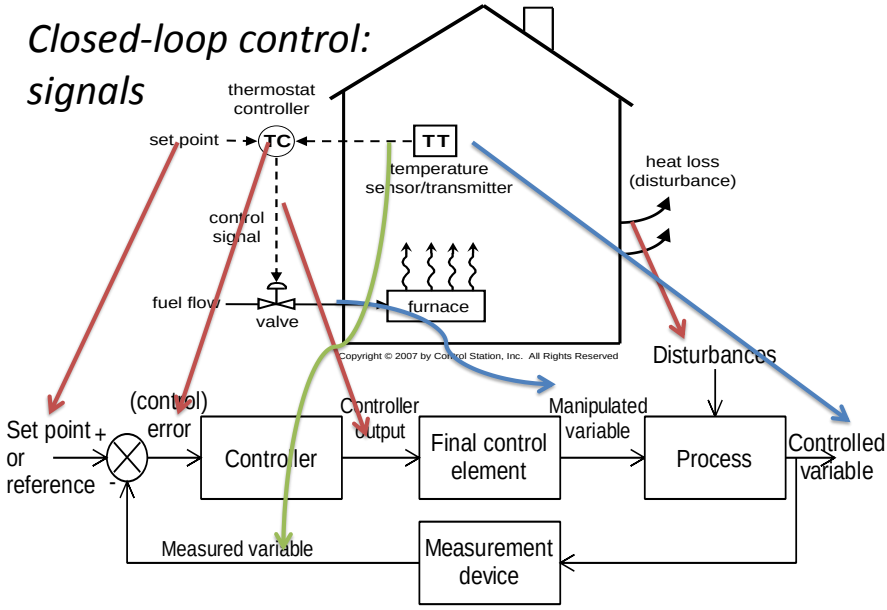
Process



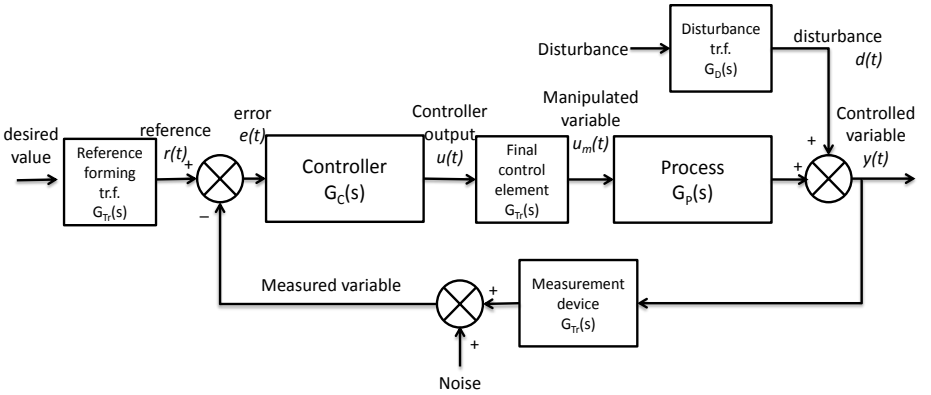
Closed-loop control: elements



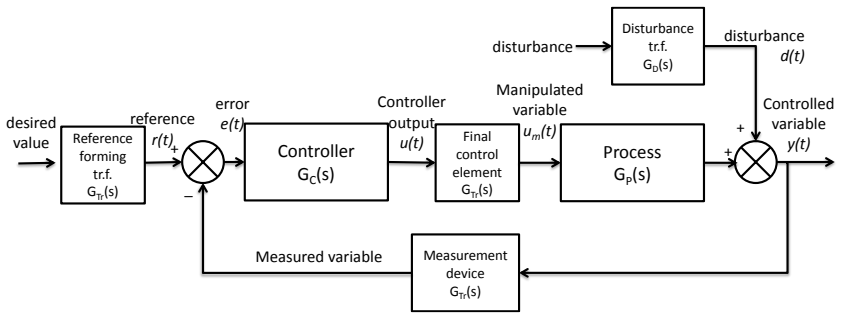
Closed-loop control: signals

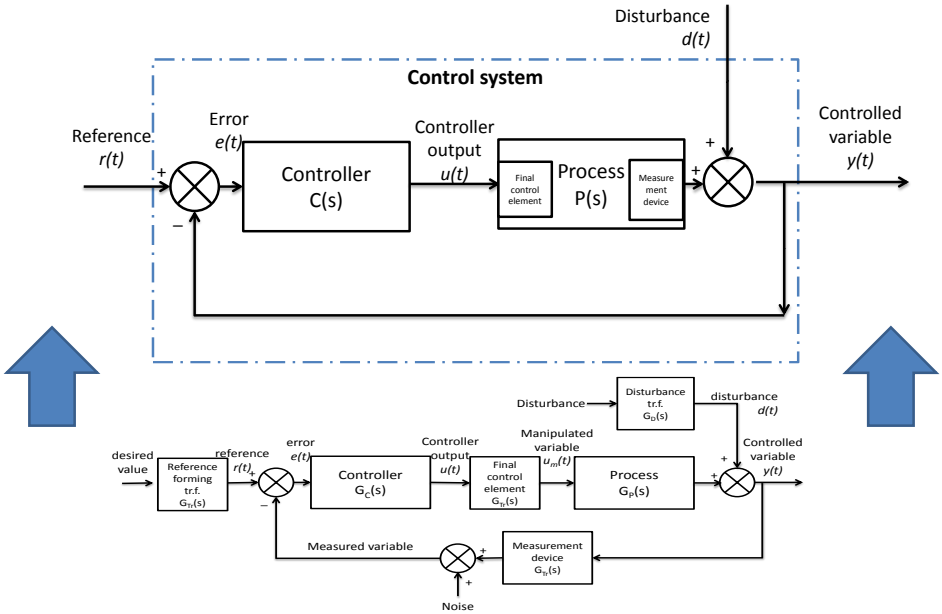


Control system model with disturbance and noise consideration

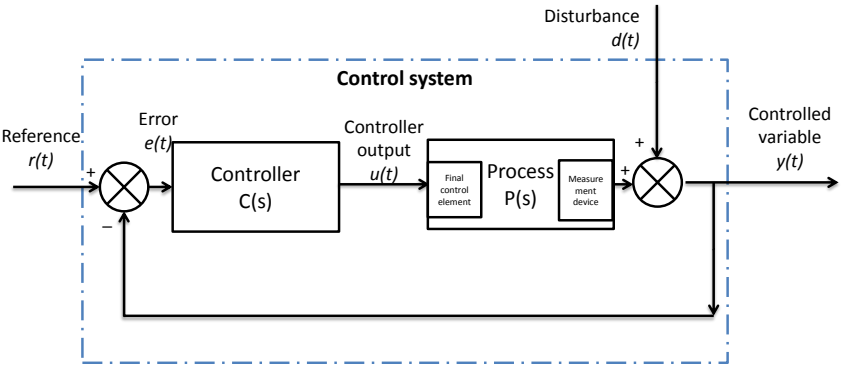


Control system model without high frequency noise consideration

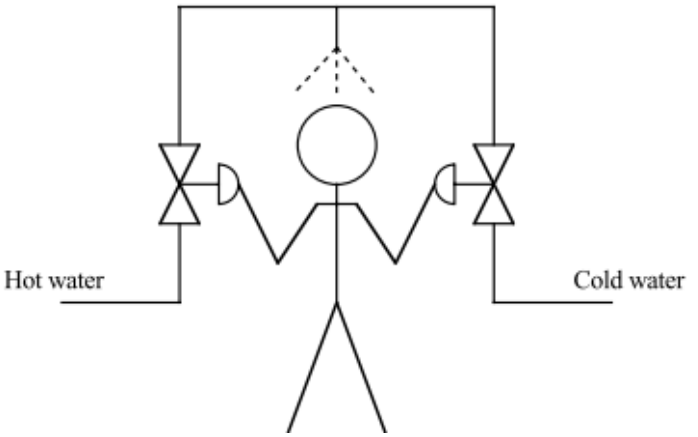




Simplified control system model: two inputs and one output



Open or closed-loop conrol?

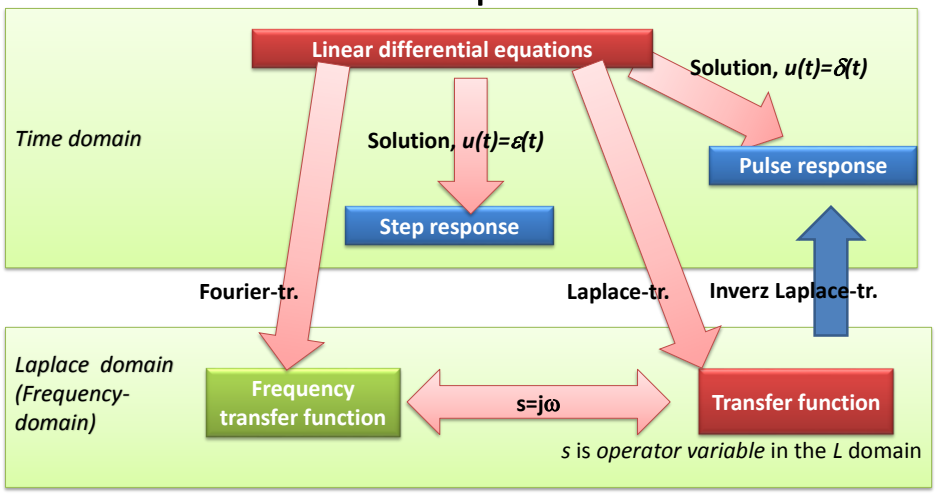


Basic system types

Our investigation will be limited to the control of dynamic, linear, single-variable, constant parameter systems. These systems can be described by the next four methods:

- n-th order linear differential equations with constant coefficients
- transfer function, frequency transfer function
- time functions (response functions)
- state-space representation is also possible, but principles of classic control systems are completely understandable by the first three modes of descriptions.

System descriptions in Time and Laplace domains

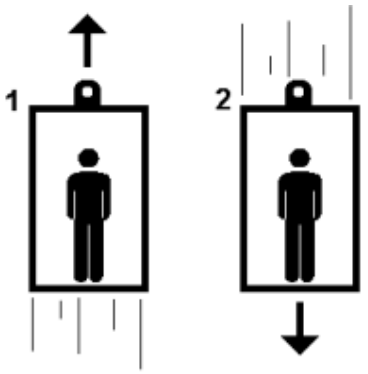


Analysis of control systems

The main objective of a control system is to produce a desired system, reducing errors and achiving system's stability

What do we analyze in control system?

Transient (temporary) response	Steady-state response	Stability

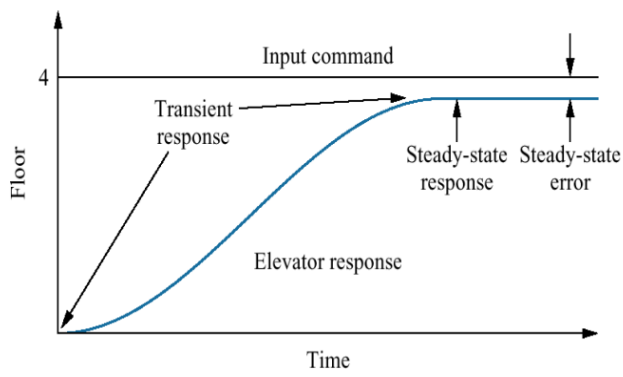


Transient response

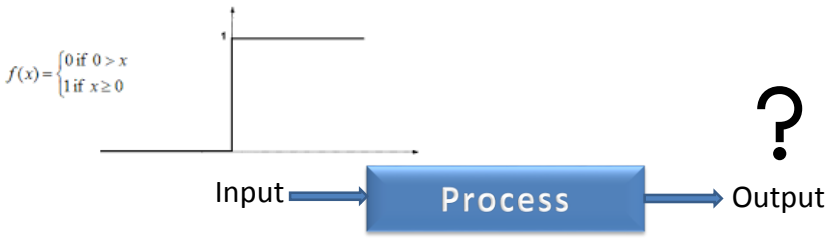
- Also known as the natural response
- Example: consider an elevator going from the first floor to the fourth floor
- If a transient response is:
  - Too slow – passengers would be angry
  - Too fast – you would be scared

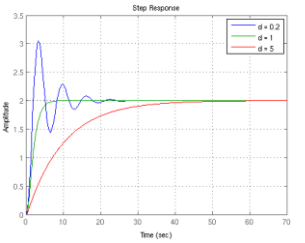
Steady-state response

- An approximation to the desired response
- It is also the response that exist for a long time following the given input signal
- In the previous lift example, the steady-state response is when the lift is about to reach the fourth floor (Nise)

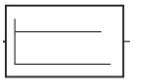
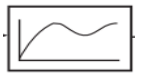
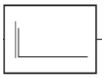


Step response based system model approximations





Basic system types

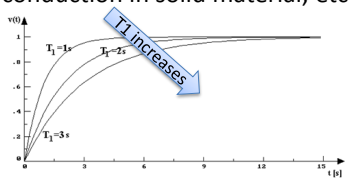
System type	Step response
Proportional	
Proportional, first-order	
Proportional, second-order	
Integrator	
Differenciator	
Time delay, Transport lag or Dead time	

Proportional, first-order system

Proportional, first-order system: PT_1	
Differential equation	$T_1\dot{y}(t) + y(t) = ku(t)\ \ y(0) = 0$
Parameter	T_1: Time constant, k: gain
Step response	$v(t) = k(1 - e^{-\frac{t}{T_1}})$
Step response: steady-state	$v_{ss}(t) = k\varepsilon(t)$
Step response: transient	$v_{tr}(t) = -ke^{-\frac{t}{T_1}}$
Pulse response	$g(t) = \frac{k}{T_1}e^{-\frac{t}{T_1}}$
Transfer function	$G(s) = \frac{k}{T_1 s + 1}$

Proportional, first-order system

One energy storage tank:
(RC-tag, free-flow tank, perfectly mixed container, diffusion, heat conduction in solid material, etc.)
Typical system's parameters : k, T_1



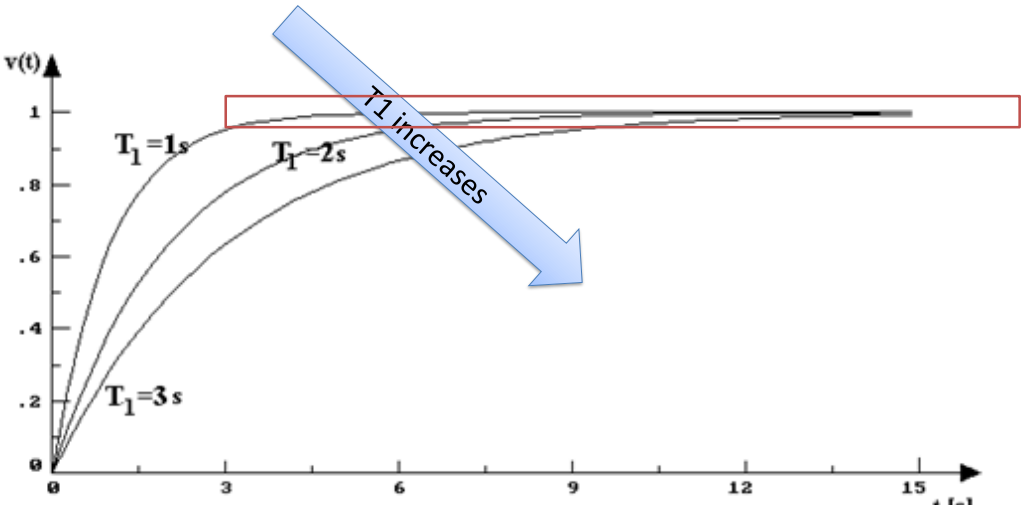
Step response : $v(t) = k(1 - e^{-\frac{t}{T_1}})$
Steady-state response : k . $(t \to \infty)$
$When\ t = T_1$ the transient component is

$$-ke^{-\frac{T_1}{T_1}} = -\frac{k}{e} = -0{,}367k$$

and the step respons value is

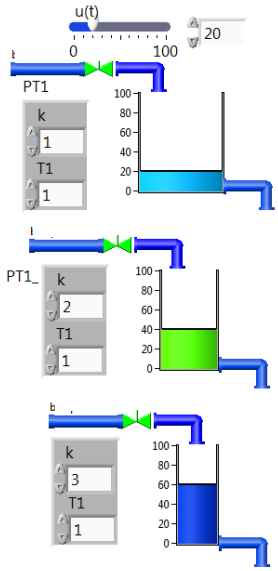
$$k - 0{,}366k = k(1 - 0{,}366) = \mathbf{0,633}\boldsymbol{k}$$

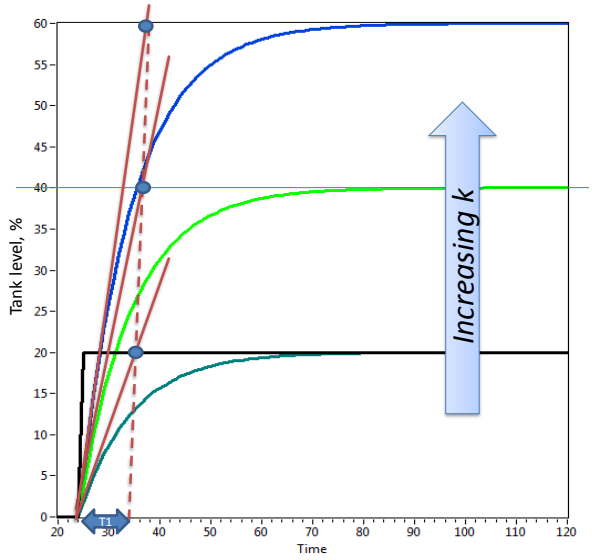
T1 changes, gain constant: *k=1*



by raising *T1* less steep rise, slower to the new equilibrium situation.

k changes, T_1=*constant, input flow: 0→20% at t=24*





First-order plus time delay model

The transfer function of a first-order plus dead time or time delay (FOPDT) model is:

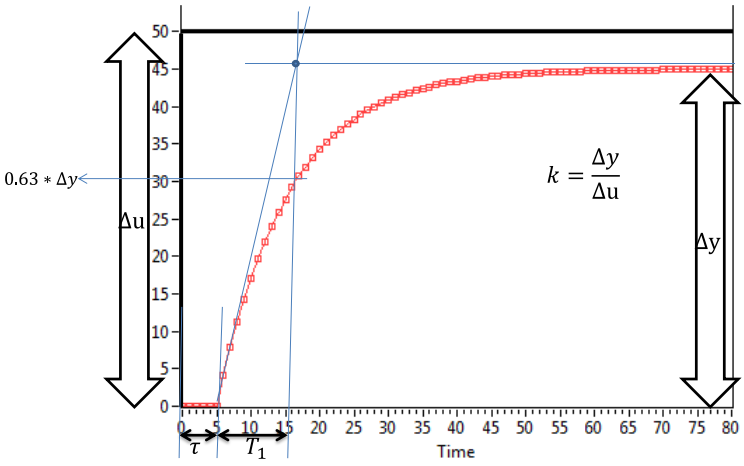
$$P(s) = \frac{k}{T_1 s + 1} e^{-\tau s}$$

The model parameters are *k:* gain, T_1 : process time constant and τ time delay

The ***time delay*** is also known as ***dead time***.

- Dead-Time is the Killer of Control: tight control grows increasingly difficult as τ becomes large
- The process time constant is the clock of the process. Dead time is large or small relative only to T_1
- When dead time grows such that $\tau > T_1$, model predictive control strategies such as a Smith predictor may show benefit

Dead time:
Step response of a FOPDT process



The FOPTD Model Parameters

In Summary

For a change in controller output:

- Process Gain, k → How Far controlled variable travels
- Time Constant, T_1 → How Fast controlled variable responds
- Dead Time, τ → How Much Delay Before controlled variable responds

Second order systems

$$T_2^2\ddot{y}(t) + T_1\dot{y}(t) + y(t) = ku(t)$$

$$T_2 = T$$

$$\xi = \frac{1}{2}\frac{T_1}{T_2}$$

$$T_2{}^2\ddot{y}(t) + 2\xi T_2\dot{y}(t) + y(t) = ku(t)$$

$$\omega_0 = \frac{1}{T}$$

$$\frac{1}{\omega_0}\ddot{y}(t) + \frac{2\xi}{\omega_0}\dot{y}(t) + y(t) = ku(t)$$

$$\xi: damping\ ratio$$

$$\omega_0: natural\ frequency$$

Typical system parameters

Gain, k

Time constant (T_2) or Natural frequency, ω_0

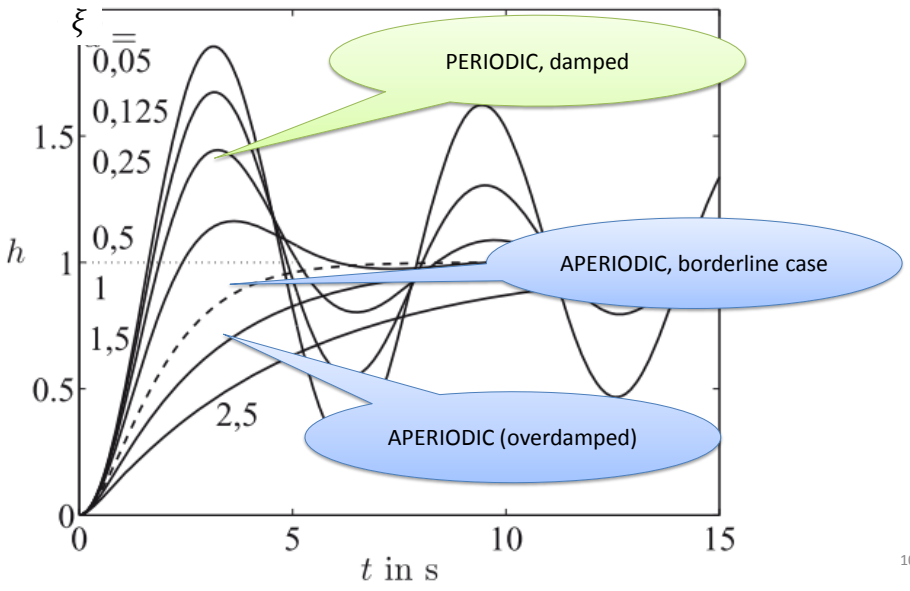
Damping ration, ξ

$$T_2 = \frac{1}{\omega_0}$$

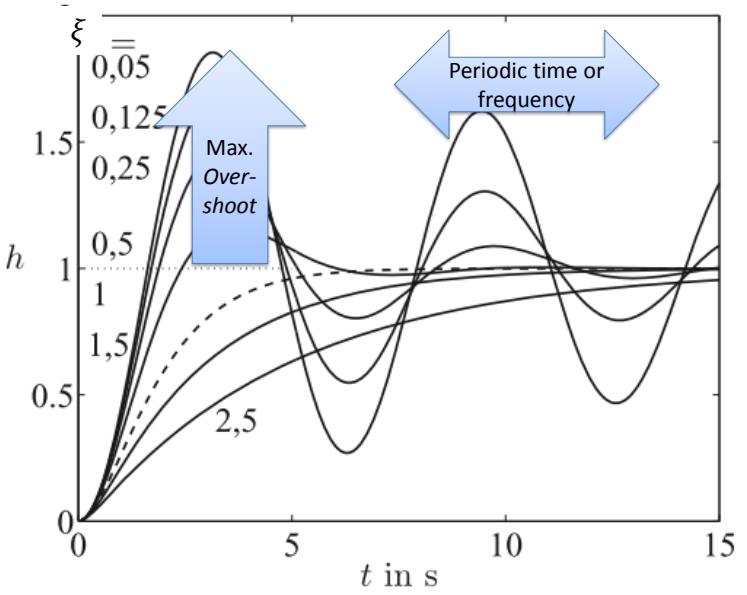
Transfer function

$$G(s) = \frac{k}{T_2{}^2 s^2 + 2\xi T_2 s + 1}$$

Transient properties

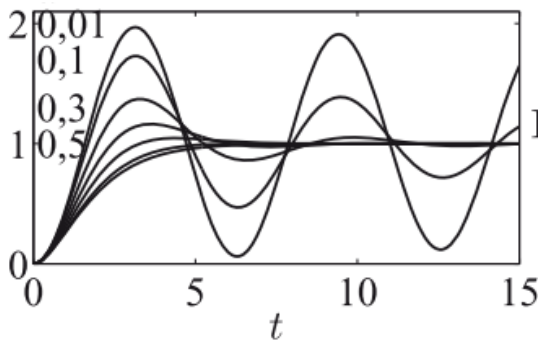


Transient properties

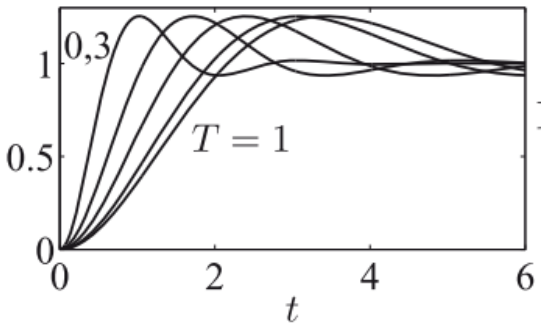


The damping ratio

1. $\xi > 1$: aperiodic step response: consists of two serial connected first order system
2. $\xi = 1$: aperiodic borderline case
3. $0 < \xi < 1$ step response with damped oscillation,
4. $\xi = 0$ oscillation with constant amplitude



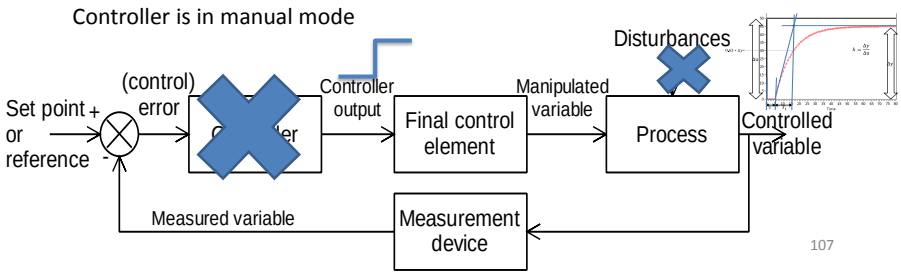
Time constant or natural frequency change, (ξ=0.4)



Increasing T „stretches" the transient, the oscillation tendency is not affected

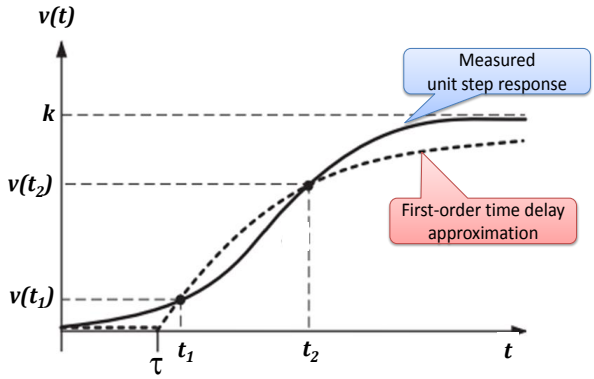
Step response based system model approximations

- Determine the step response function of the system for a jump signal
- Looking for a simpler system model whose response function approximates the measured transition
- Approximation is often done graphically.



Model reduction to FOPTD model

The model reductions means that the higher-order system characteristic will be approximated by a system with same gain an apparent time delay and dominant time constant.

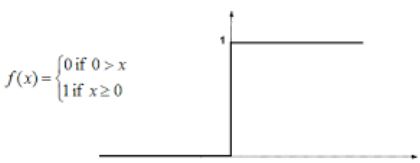


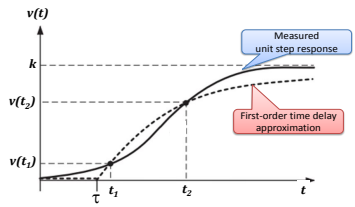
Step response based system model approximations

Procedure:

- Determine the step response function of the system for a jump signal
- Looking for a simpler system model whose response function approximates the measured transition

Approximation is often done graphically





Model reduction to FOPTD model

The transfer function of a FOPTD process:

$$P(s) = \frac{k}{T_1 s+1} e^{-\tau s}$$

Parameters must be determined:

k gain,

T_1 time constant

τ time delay.

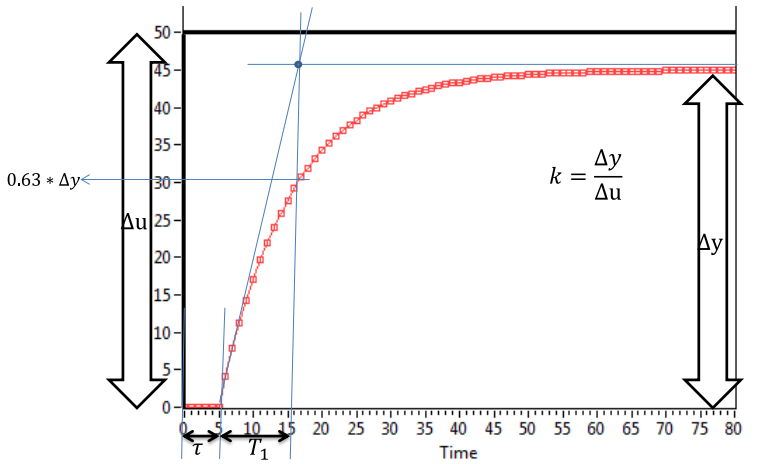
Step response:

$$v(t) = k\left(1 - e^{-\frac{t-\tau}{T_1}}\right)$$

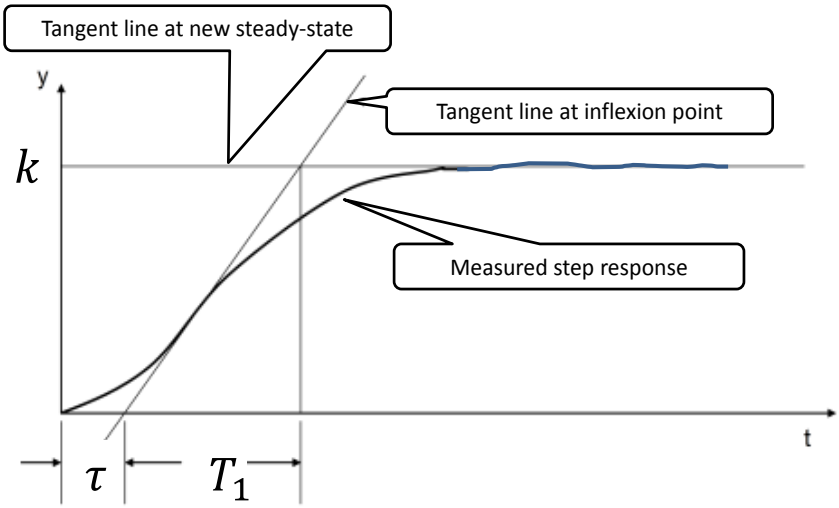
Differential equation:

$$T_1 \dot{y}(t) + y(t) = ku(t-\tau),\ y(0)=0$$

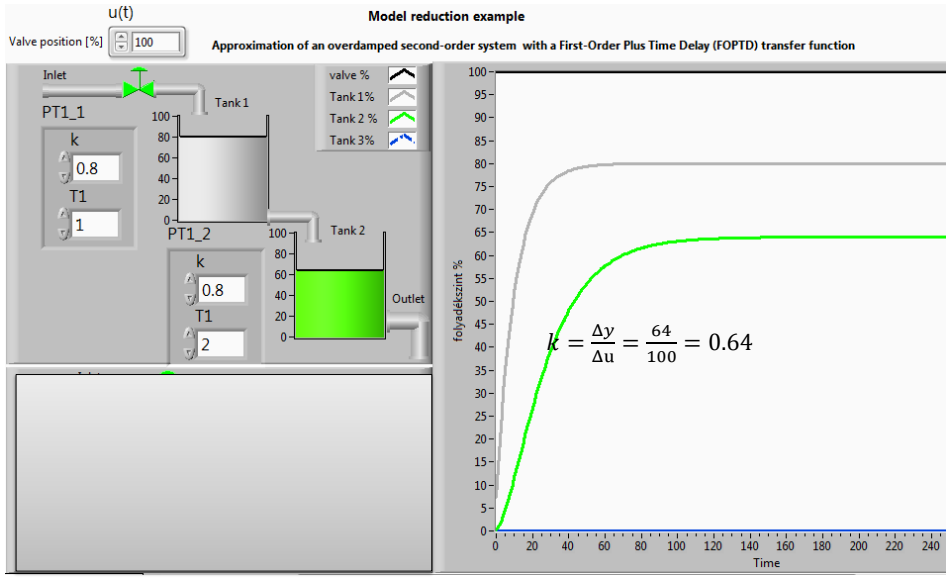
Graphic presentation of the system parameters in the step response of a FOPTD process



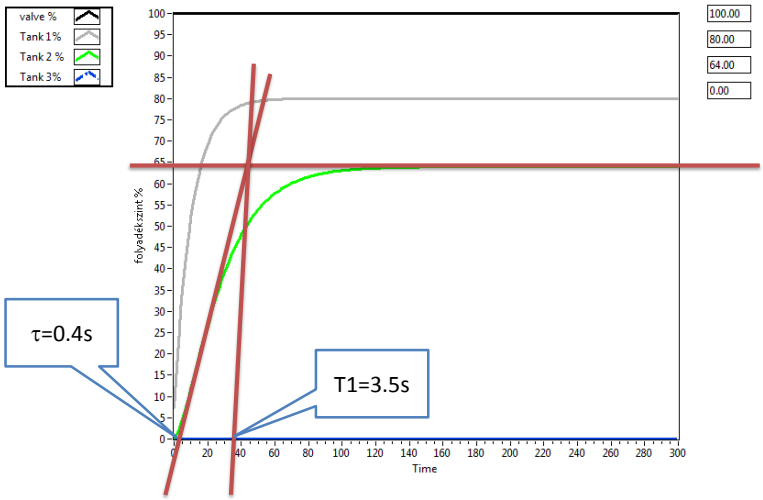
1. Graphic method: Slope-intercepts method



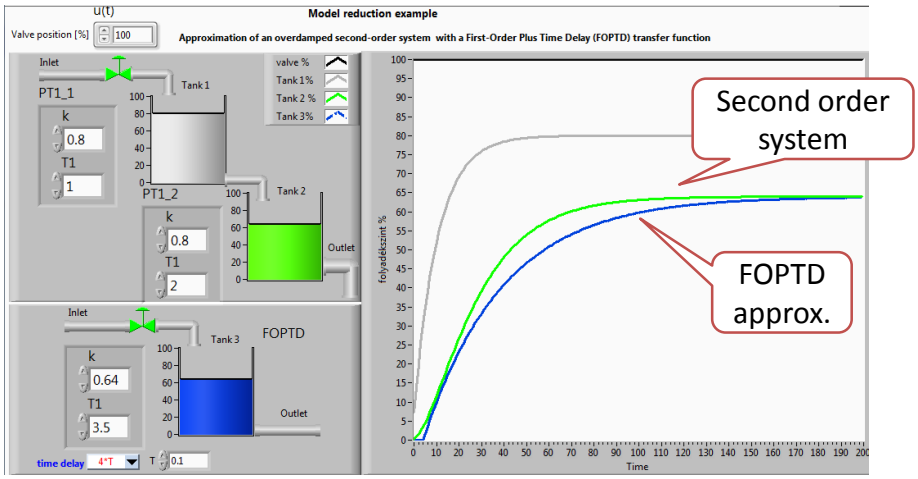
Example: FOPTD approximation of a second order system



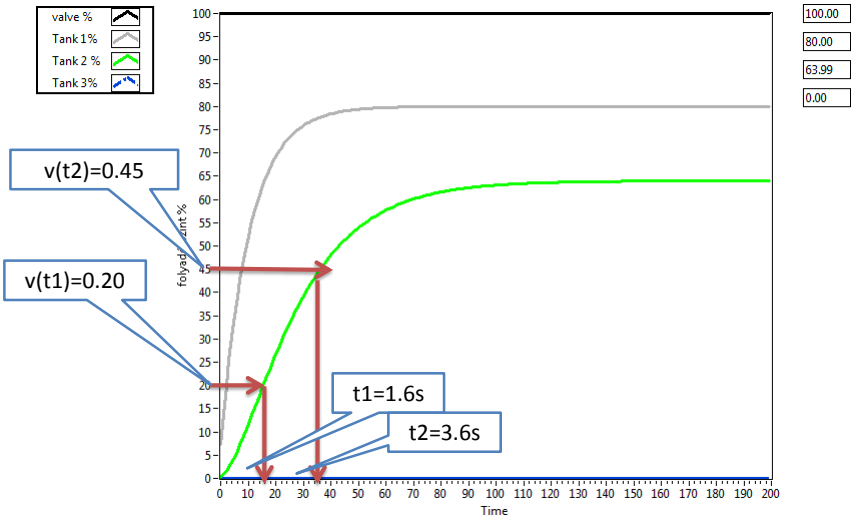
Applying the Slope-intercepts method



Testing



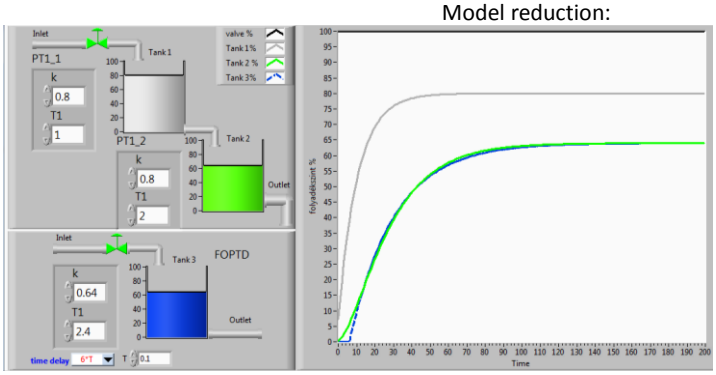
Two points method



Calculation of parameters

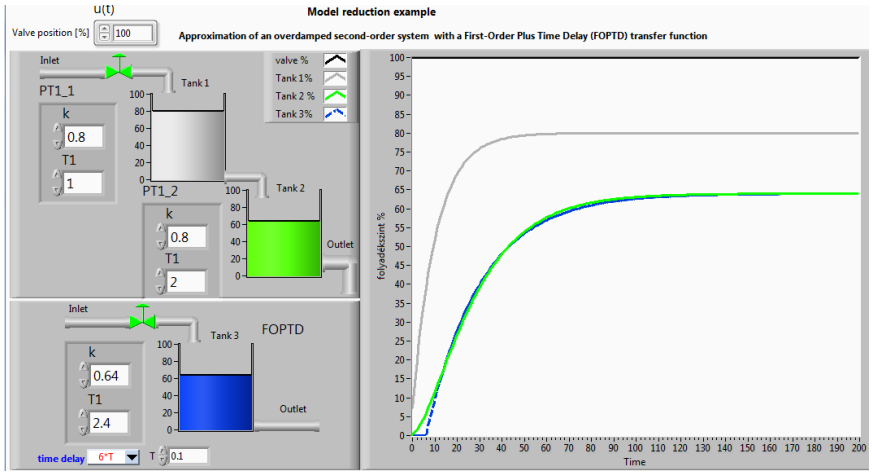
$$v(t_1) = k\left(1 - e^{-\frac{t_1-\tau}{T_1}}\right),$$
$$v(t_2) = k\left(1 - e^{-\frac{t_2-\tau}{T_1}}\right), \quad \tau = T_1 ln\left(1 - \frac{v(t_1)}{k}\right) + t_1$$
$$\tau = T_1 ln\left(1 - \frac{v(t_2)}{k}\right) + t_2, \quad T_1 = \frac{t_2 - t_1}{ln\left(\frac{k - v(t_1)}{k - v(t_2)}\right)}$$

Model reduction:



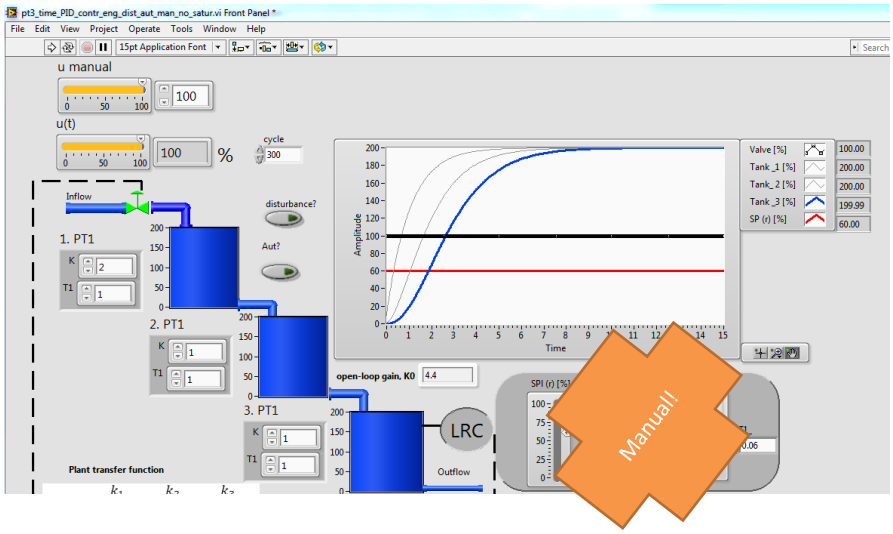
$$0.20 = 0.64\left(1 - e^{-\frac{1.6-\tau}{T_1}}\right)$$
$$0.45 = 0.64\left(1 - e^{-\frac{3.6-\tau}{T_1}}\right)$$
$$\tau = T_1 ln\left(1 - \frac{v(t_1)}{k}\right) + t_1$$
$$\tau = T_1 ln\left(1 - \frac{v(t_2)}{k}\right) + t_2$$
$$T_1 = \frac{3.6 - 1.6}{ln\frac{0.64 - 0.2}{0.64 - 0.45}}$$
$$T_1 = \frac{3.6 - 1.6}{ln\frac{0.64 - 0.2}{0.64 - 0.45}} = \frac{2}{ln\frac{0.44}{0.19}} = 2.4s$$
$$\tau = T_1 ln\left(1 - \frac{v(t_1)}{k}\right) + t_1$$
$$\tau = 2.4 ln\left(1 - \frac{0.2}{0.64}\right) + 1.6$$
τ =-0.9+1.5=0. 6s

Teszt

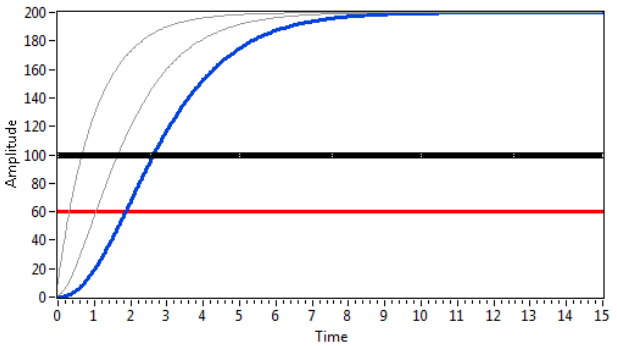


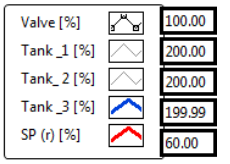
Example : PI-control of a third-order process

1. Step respons measurement

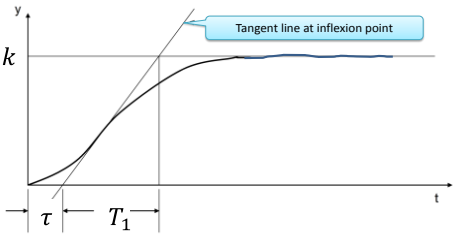


Approximation: FOPDT

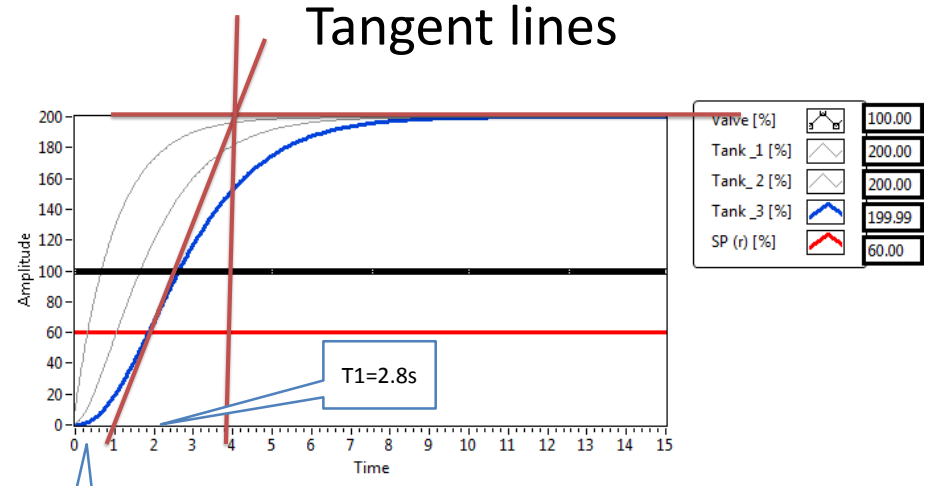




Measure the third-order system step response and approximate with a FOPDT model



Tangent lines



τ=1s

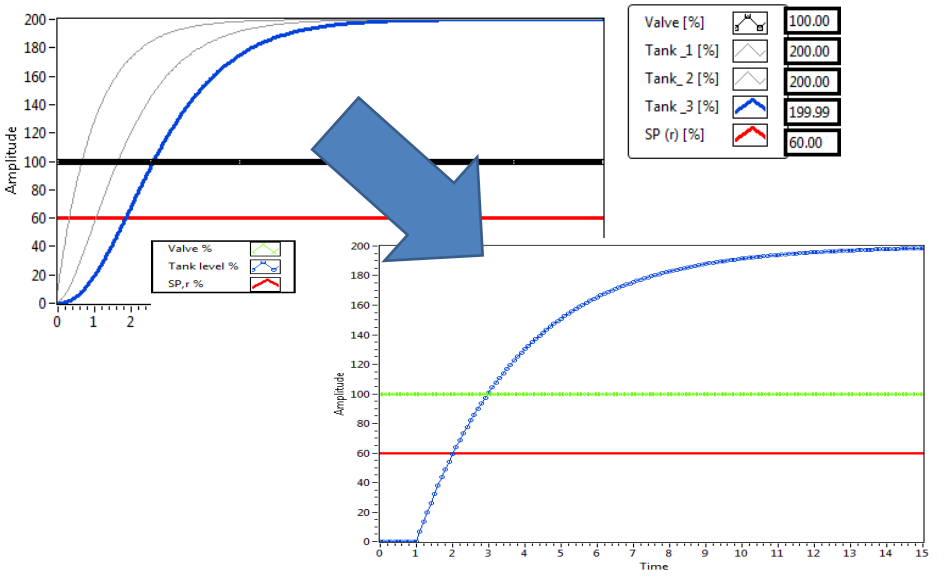
$$k = \frac{\Delta y}{\Delta u} = \frac{200}{100} = 2$$

The FOPDT model

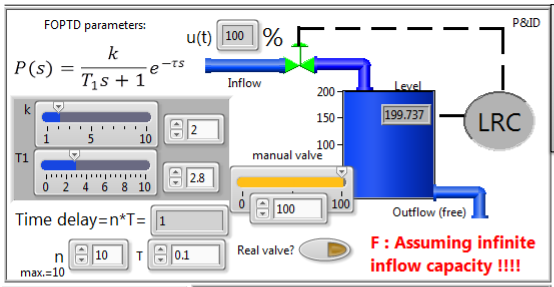
- *General model*:
- $P(s) = \frac{k}{T_1 s+1} e^{-\tau s}$
- *In our case:*
- $P(s) = \frac{2}{2.8s+1} e^{-s}$

- Instead of $P(s) = \frac{2}{(s+1)^3}$!

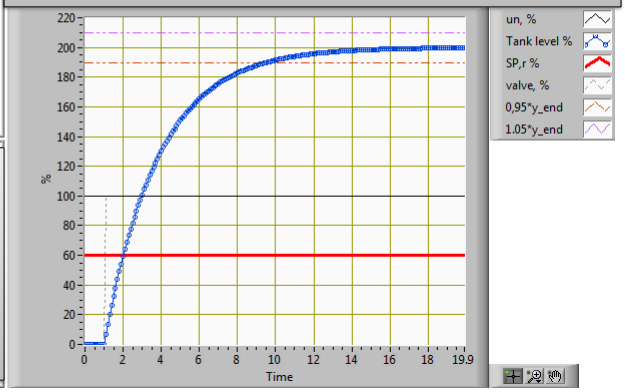
Approximation result



Test the step response



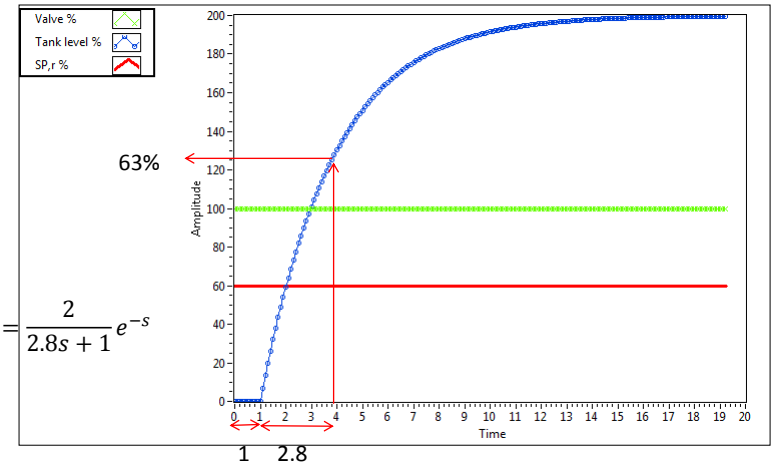
$$P(s) = \frac{2}{2.8s + 1} e^{-s}$$



Set the process parameters (k, T1 and delay time)
Controller in manual mode
Open valve manually to 100%
Run the program

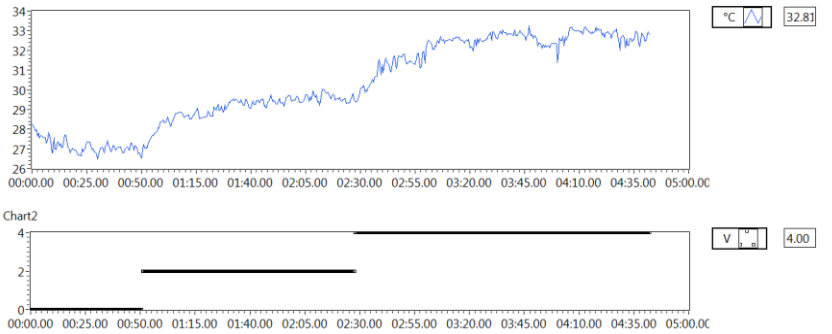
Step response of the FOPDT modell

$$P(s) = \frac{2}{2.8s + 1} e^{-s}$$



Example: HVAC model, process temperature step responses

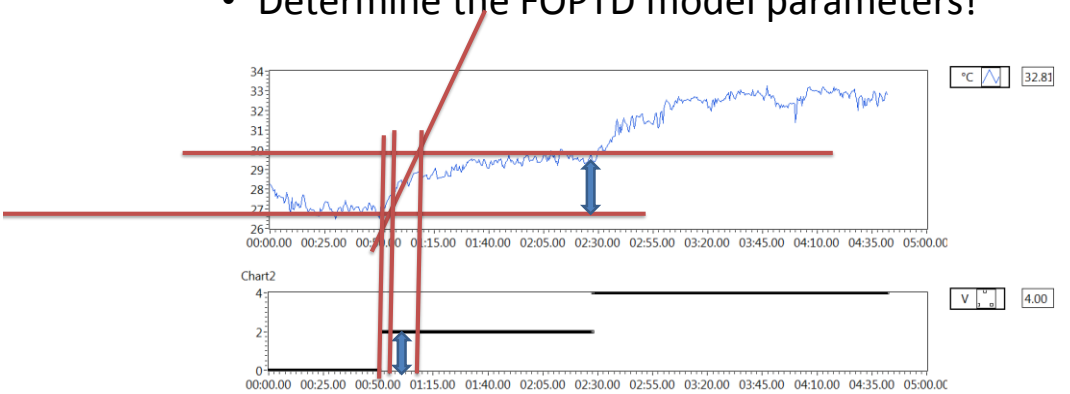
- Determine the FOPTD model parameters!



With noisy and scattered data, the precision of fitting a higher-order plus dead-time model is low this is one reason why many of the empirical controller design methods are based on the simpler first order plus time delay function.

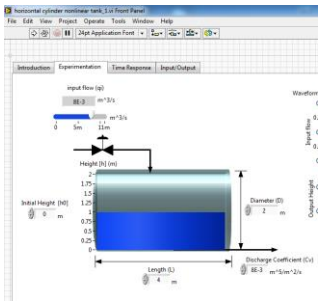
Example: HVAC model, process temperature step responses

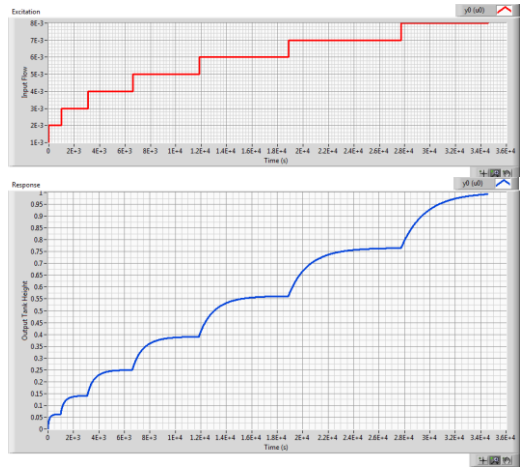
- Determine the FOPTD model parameters!



Processes have Nonlinear Behavior!

- A FOPTD model response is constant as operating level changes
- **Since the FOPTD model is used for controller design and tuning, a process should be modeled at a specific design level of operation!**





Intelligent control systems

INTRODUCTION TO PROGRAMMABLE CONTROLLERS

Introduction to Programmable Controllers

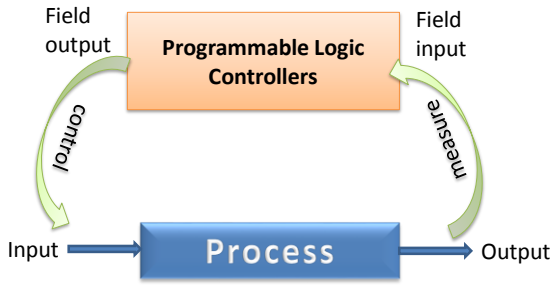
Every aspect of industry—from power generation to automobile painting to food packaging—uses programmable controllers to expand and enhance production.

Programmable Logic Controllers or *PLCs*, are **solid-state** members of the computer family, using integrated circuits instead of electromechanical devices to implement control functions.

They are capable of storing instructions, such as sequencing, timing, counting, arithmetic, data manipulation, and communication, to control industrial machines and processes.

PLC conceptual application diagram

PLCs can be thought of in simple terms as industrial computers with specially designed architecture in both their central units (the PLC itself) and their interfacing circuitry to field devices (input/output connections to the real world). Their design roots based on the principles of simplicity and practical application.



Historical background

General Motors Corporation specified the design criteria for the first programmable controller in 1968.

The new control system had to be price competitive with the use of relay systems.

The system had to be capable of sustaining an industrial environment.

The input and output interfaces had to be easily replaceable.

The controller had to be designed in modular form, so that subassemblies could be removed easily for replacement or repair.

The control system needed the capability to pass data collection to a central system.

The system had to be reusable.

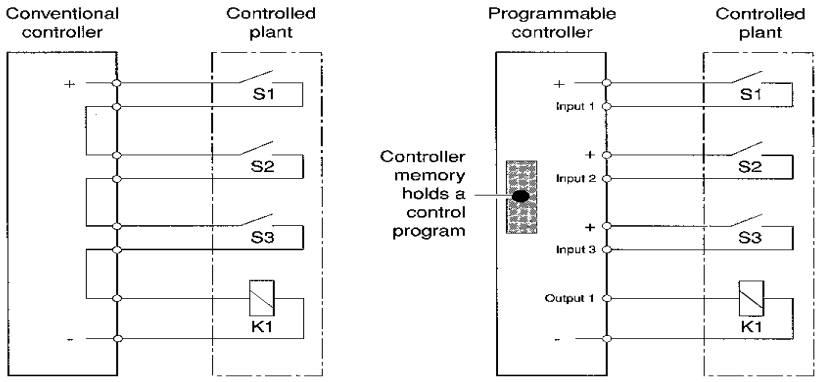
The method used to program the controller had to be simple, so that it could be easily understood by plant personnel.

In a short period, programmable controller use started to spread to other industries. By 1971, PLCs were being used to provide relay replacement as the first steps toward control automation in other industries, such as food and beverage, metals, manufacturing, and pulp and paper.

Historical background

The first PLCs offered relay functionality, thus replacing the original hardwired **relay logic**, which used electrically operated devices to mechanically switch electrical circuits. They met the requirements of modularity, expandability, programmability, and ease of use in an industrial environment.

These controllers were easily installed, used less space, and were reusable.

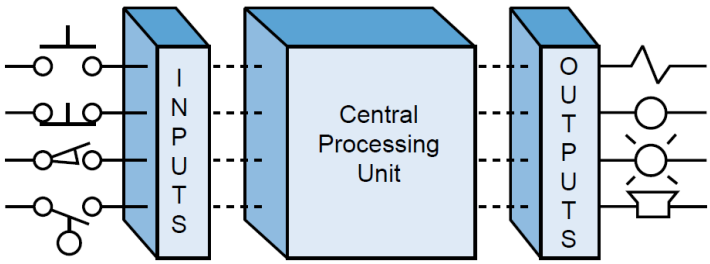


Principles of operation

A programmable controller consists of two basic sections:

the central processing unit

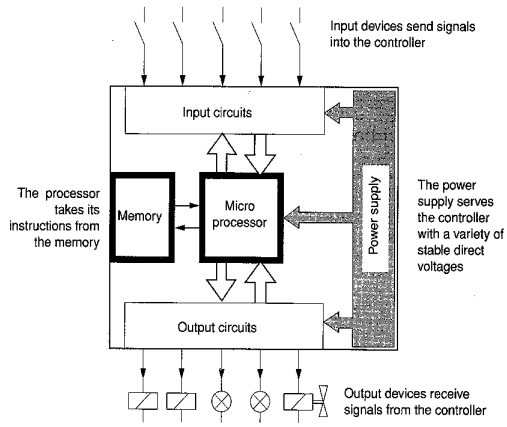
the input/output interface system



Principles of operation: the CPU

The central processing unit (CPU) governs all PLC activities.
The following three components form the CPU:

- the processor
- the memory system
- the system power supply



The input/output system

The **input/output (I/O) system** is physically connected to the field devices that are encountered in the machine or that are used in the control of a process. These field devices may be discrete or analog input/output devices, such as limit switches, pressure transducers, push buttons, motor starters, solenoids, etc.

The I/O interfaces provide the connection between the CPU and the information providers (inputs) and controllable devices (outputs).

During its operation, the CPU completes three processes:

(1) it **reads**, or accepts, the input data from the field devices via the input interfaces,

(2) it **executes**, or performs, the control program stored in the memory system, and

(3) it **writes**, or updates, the output devices via the output interfaces.

This process of sequentially reading the inputs, executing the program in memory, and updating the outputs is known as **scanning.**

Graphical representation of a scan

input

↓

memory area

↓

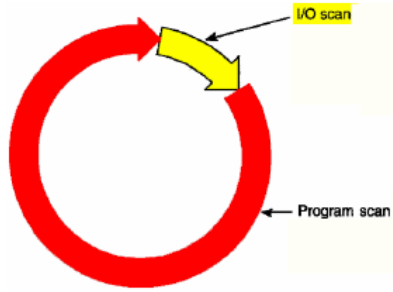
program

↓

memory area

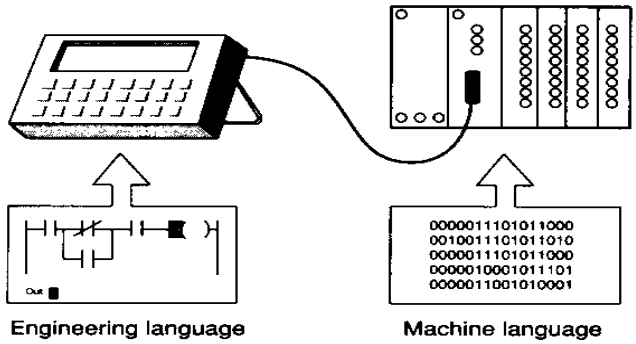
↓

output

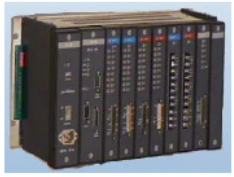


The **programming device**

usually a personal computer or a manufacturer's miniprogrammer unit is required to enter the control program into PLC memory



PLCs Versus Personal Computers

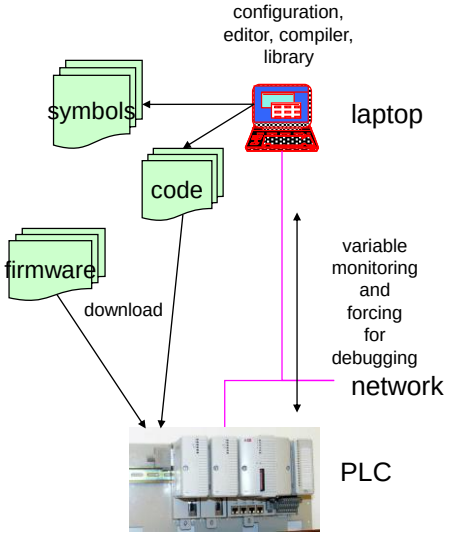


Same basic architecture

PLC
- Operates in the industrial environment
- Is programmed in relay ladder logic
- Has no keyboard, CD drive, monitor, or disk drive
- Has communications ports, and terminals for input and output devices

PC
- Capable of executing several programs simultaneously, in any order
- Some manufacturers have software and interface cards available so that a PC can do the work of a PLC

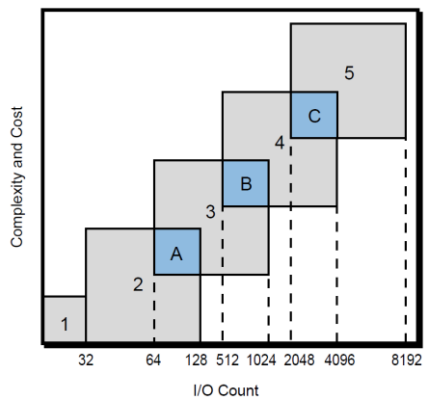
IEC 61131 Programming environment



PLC product ranges

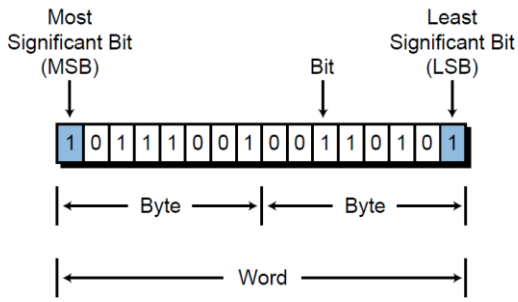
The PLC market can be segmented into five groups:

1. micro PLCs
2. small PLCs
3. medium PLCs
4. large PLCs
5. very large PLCs



Registers: One word, two bytes, sixteen bits

The microprocessors used in PLCs are categorized according to their word size, or the number of bits that they use simultaneously to perform operations.
Standard word lengths are 8, 16, and 32 bits. This word length affects the speed at which the processor performs most operations.



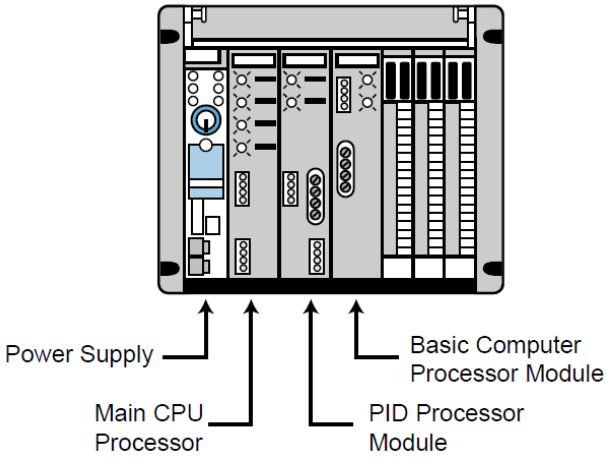
A multiprocessor configuration

The CPU of a PLC system may contain more than one processor (or micro) to execute the system's duties and/or communications, because extra processors increase the speed of these operations. This approach of using several microprocessors to divide control and communication tasks is known as **multiprocessing**.

Another multiprocessor arrangement takes the microprocessor intelligence away from the CPU, moving it to an intelligent module. This technique uses intelligent I/O interfaces, which contain a microprocessor, built-in memory, and a mini-executive that performs independent control tasks.

Typical intelligent modules are proportional-integral-derivative (PID) control modules, which perform closed-loop control independent of the CPU, and some stepper and servo motor control interfaces.

A multiprocessor configuration



The scan time

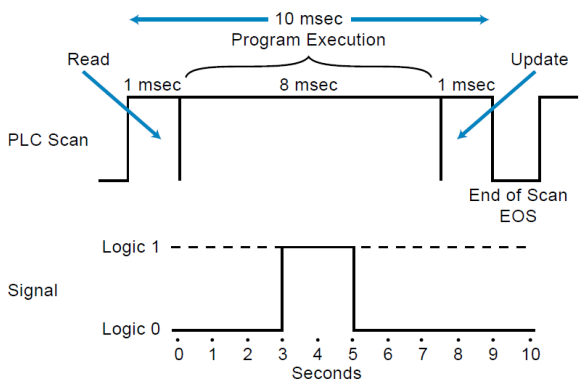
- The time it takes to implement a scan is called the **scan time**. The scan time is the total time the PLC takes to complete the program and I/O update scans.
- The program scan time generally depends on two factors:
- (1) the amount of memory taken by the control program and
- (2) the type of instructions used in the program (which affects the time needed to execute the instructions).
- The time required to make a single scan can vary from a few tenths of a millisecond to 50 milliseconds.

The scan time

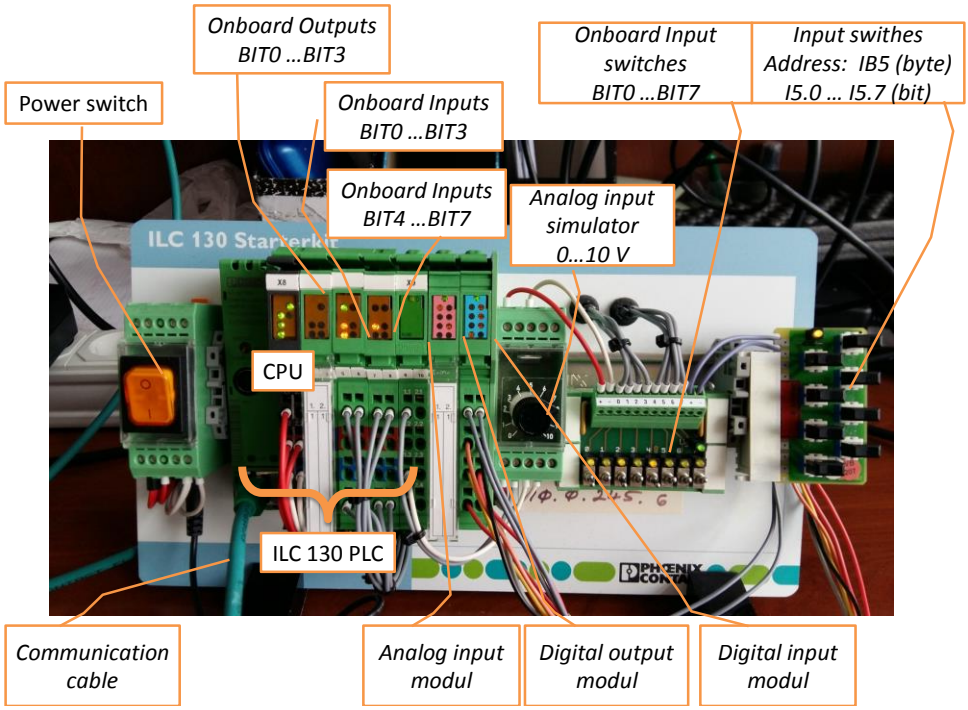
- PLC manufacturers specify the scan time based on the amount of application memory used (e.g., 1 msec/1K of programmed memory).
- However, other factors also affect the scan time. The use of remote I/O subsystems can increase the scan time, since the PLC must transmit and receive the I/O update from remote systems.
- Monitoring control programs also adds time to the scan, because the microprocessor must send data about the status of the coils and contacts to a monitoring device (e.g., a PC).

A processor is able to read an input as long as the input signal is not faster than the scan time

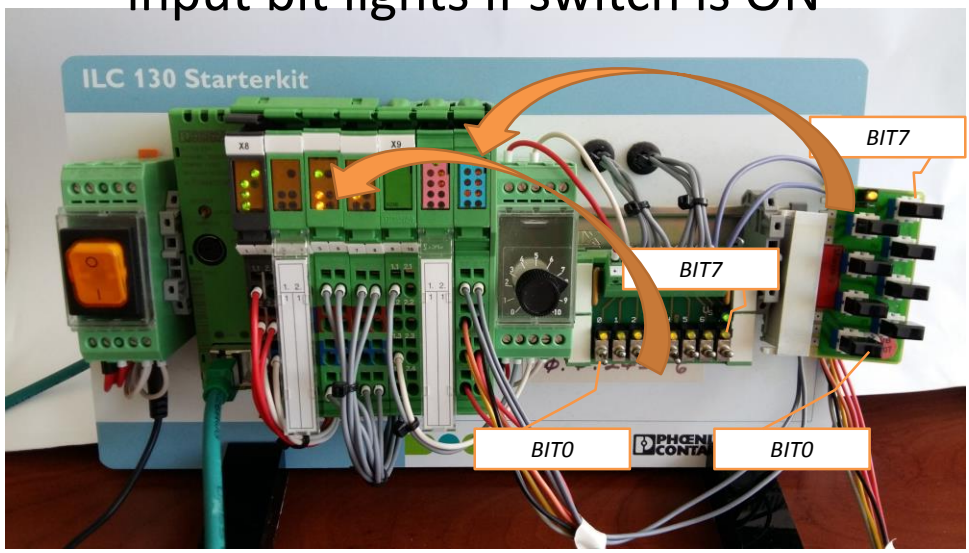
- Illustration of a signal that will not be detected by a PLC during a normal scan.



PLC configuration in lab: ILC130 Starter kit



PLC Inputs and swithes
input bit lights if switch is ON



PLC outputs light if they are true

Program runs: this LED continuously lights up

PC WORX software
with PHOENIX ILC130 PLC-s

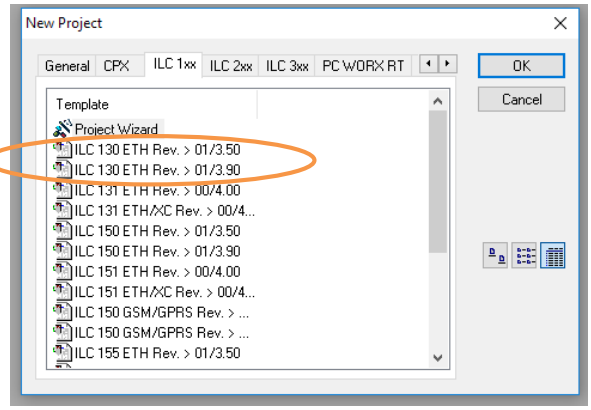
- PC Worx is the integrated programming tool for Phoenix Contact controllers. In the laboratory you can develop and online test your programs on the PHOENIX ILC130 PLC with PCWORX software system.
- The PLC development environments basically contain the next windows and workspaces:
- The project tree window, which shows the software structure of the project.
- The hardware configuration window, in which the resources (CPU-s) and the expansion modules.
- The program development and debugging workspace.
- The message window.

Creating a new project

in PCWORX

1. File: Create new project

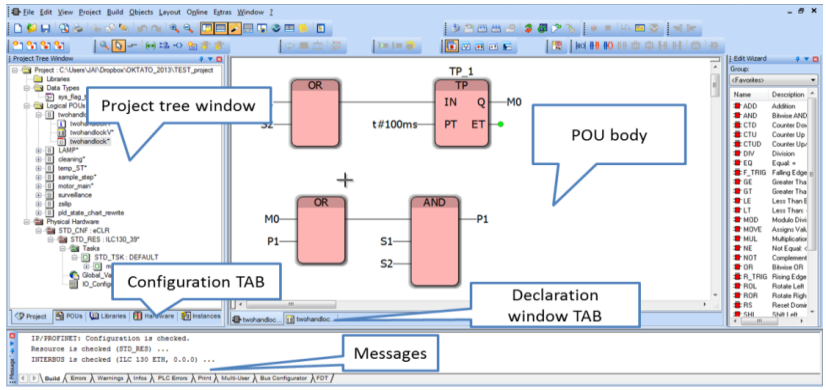
- Select the proper PLC type from the list:



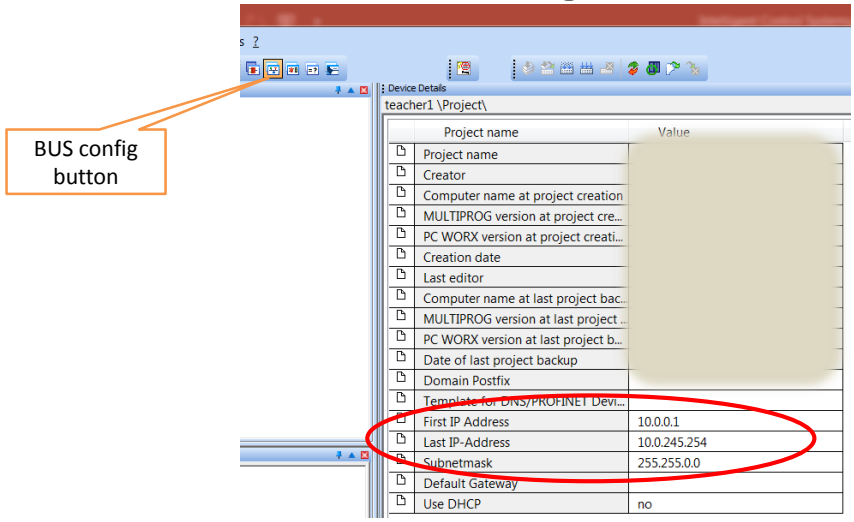
PLC with power switch: ILC 130ETH Rev.>01/3.90

PLC without power switch: ILC 130ETH Rev.>01/3.50

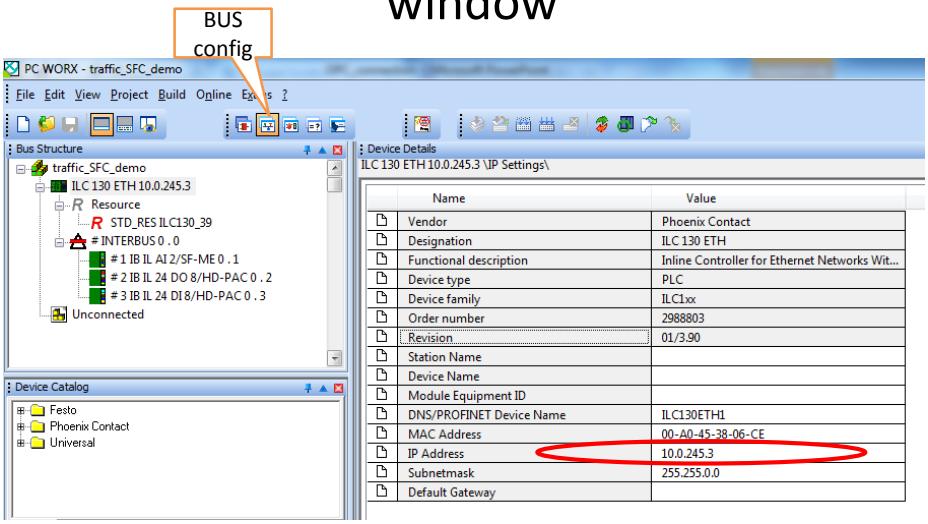
Main components of the PC WORX IEC programming workspace



2. Write First-Last IP Addresses and Subnetmask in BUS config window



3. Write PLC IP Address in BUS config window

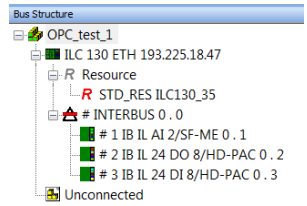


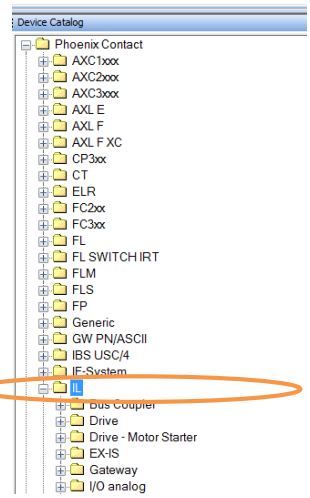
4. Define moduls

Select INTERBUS

Open in Device catalog window the Phoenix Contact directory

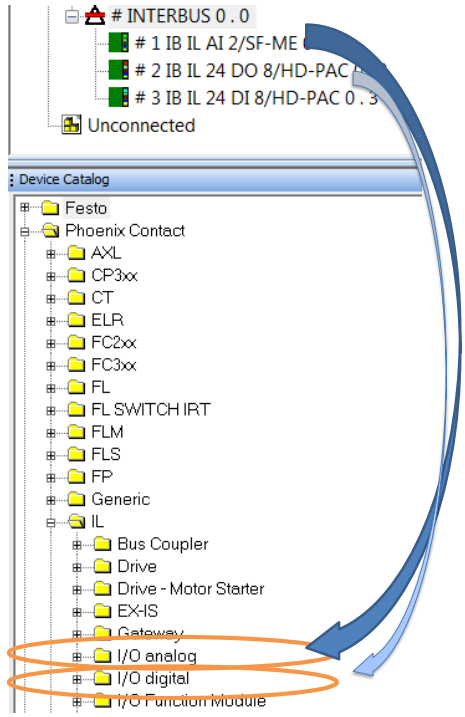
Open IL directory





BUS CONFIG

- Open the marked directories
- Select moduls from the device catalog
- See the order!

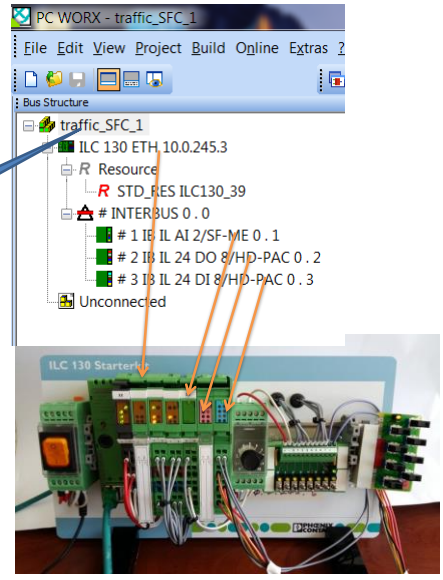


Modul declaration in order of bus places

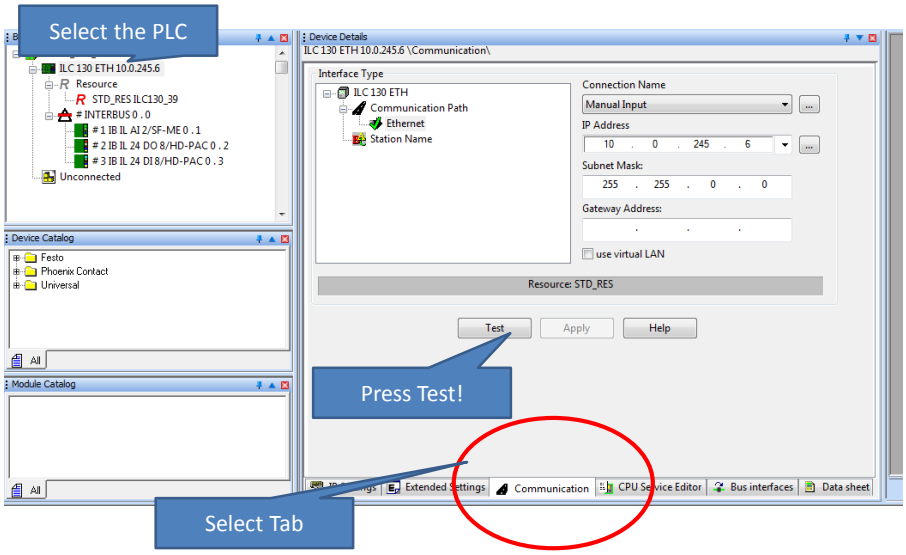
In BUS CONFIG windows:

- Save Project (As ..) !

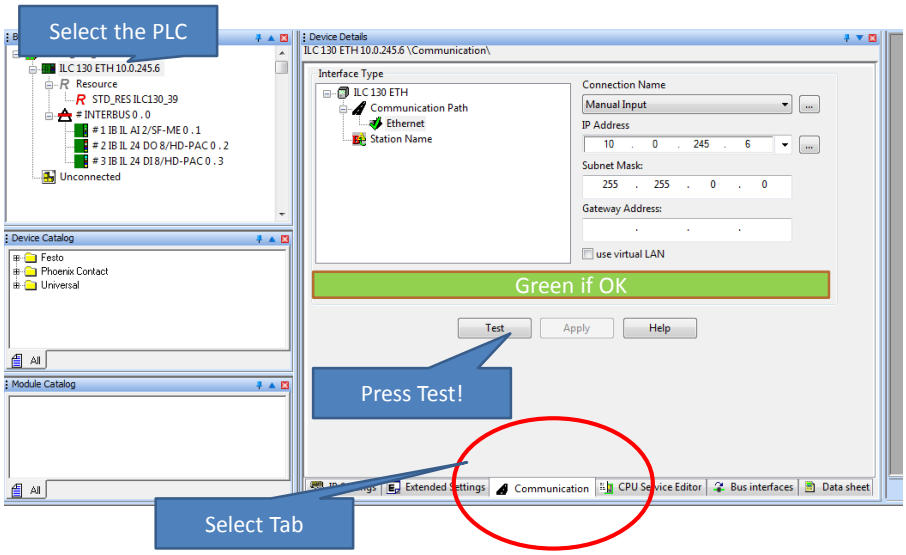
Project name



Testing communication



Testing communication

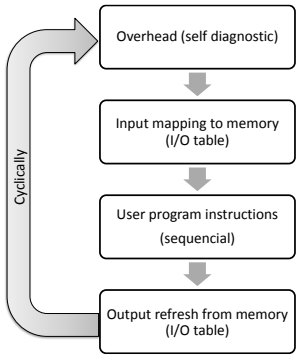


Correct configuration: green lights on top of the moduls

Intelligent control systems

CONVENTIONAL PLC PROGRAMMING ELEMENTS

The PLC program scan cycle

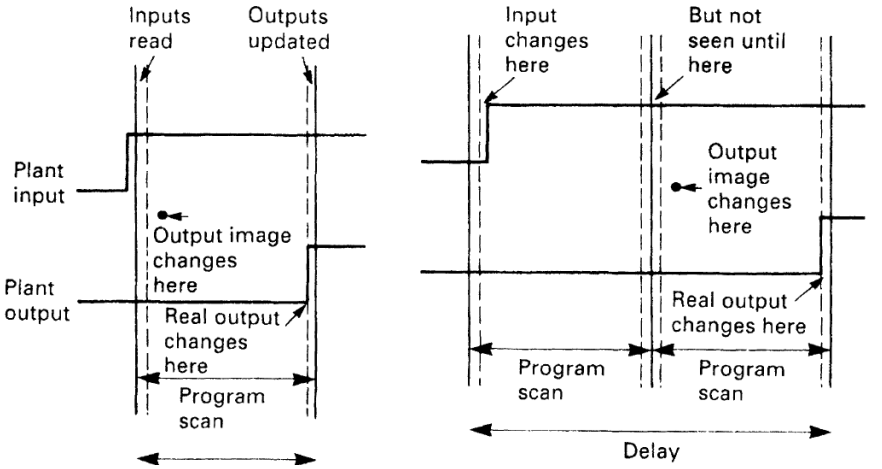


- A typical PLC program is cyclic: executed repeatedly as long as the controlled system is running.
- The status of physical input points is copied to an area of memory, called the "I/O Image Table".
- The program is then run from its first instruction to the last.
- The program execution uses only the mapped I/O-s.
- Finally the statuses of physical outputs are updated from the PLC I/O image table.
- Before the new input reading some supervisory processes are performed (overhead processes).

Cycle time or scan time vs. response time

- The *cycle time or scan time* is the time from the execution (commencement) of the I/O Refresh operation to the execution (processing) of the following I/O Refresh.
- The cycle time includes time for overhead processing (self-diagnosis), execution of user programs, I/O Refresh processing and the processing of peripheral services.
- **I/O RESPONSE TIME**
- The *I/O response time* is the time elapsed between the input change and the system replies in an output change. Response time is about double of the cycle time. It impossible to implement input changes that rate faster than the cycle time.
- Notice: the time constant (response time) of the control system must be at least one order of magnitude smaller than the smallest time constant of the controlled plant!

The effect of the program scan on the response time

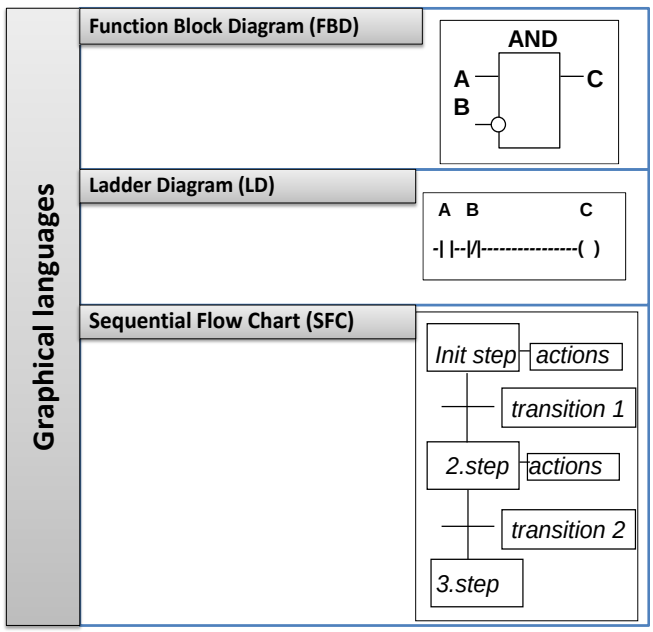


PLC programming languages

- PLC programming languages are application-oriented languages developed for cyclic execution and real-time. The languages are standardized as IEC 61131-3.
- IEC 61131 is an IEC (International Electronical Comission) standard for programmable controllers. Part 3 of IEC 61131 deals with programming languages.
- IEC 61131-3 defines two graphical and two textual PLC programming language standards:
  - Ladder diagram (LD), graphical
  - Function block diagram (FBD), graphical
  - Structured text (ST), textual
  - Instruction list (IL), textual
  - Sequential function chart (SFC), has elements to organize programs for sequential and parallel control processing.
- IEC 61131-3 is the most important automation language in industry. 80% of all PLCs support it, all new developments base on it.

textural languages examples

Textural languages	
Instruction List (IL)	**Structured Text (ST)**
LD A ANDN B ST C	C := A AND NOT B



Function Block Diagram (FBD)

Rules:

Each signal is connected to exactly one source. This source can be the output of a function block or a plant signal.

The type of the output pin, the type of the input pin and the signal type must be identical.

For convenience, the function plan should be drawn so the signals flow from left to right and from top to bottom.

The Sequential Function Chart (SFC)

The *Sequential Function Chart* (SFC) programming method is used for a pictorial representation of a system's operation to show the sequence of the events involved in its operation.

The operation is described by a number of separate sequentially connected states or steps that are represented by rectangular boxes, each representing a particular state of the system being controlled.

Each connecting line between states has a horizontal bar representing the transition condition that has to be realized before the system can move from one state to the next.

When the transfer conditions to the next state are realized, the next state or step in the program occurs.

The process thus continues from one state to the next until the entire machine cycle is completed.

Outputs/actions at any state are represented by horizontally linked boxes and occur when that state has been realized.

The Sequential Function Chart (SFC)

Rules:

there is always a transition between two states;

there is always a state between two transitions.

SFC programming implementations can be very different in various development systems.

Program Organization Units: The program

- A program has the following characteristic properties, as defined by IEC61131:
- Only the program is allowed to declare variables to be mapped to physical addresses;
- A program is allowed to call functions and instances of function blocks.

Program Organization Units: The function block

- A function block, as defined by IEC61131, has the following characteristic properties:
- It may have one, more than one, or no inputs;
- It may have one, more than one, or no outputs;
- Multiple instances can be created of a function block, and each instance will keep a private copy of all data associated with that function block (input, output, intermediate data);
- A function block cannot be called, only instances can be called.
- The function block has a `memory`, i.e. all data (input, output, local) will keep its value from one call to the next.
- On a call, it is not necessary to supply all input data; those not provided will simply keep the value from the previous call (or the default value if there was no call before).
- A function block can call functions and instances of other function blocks.

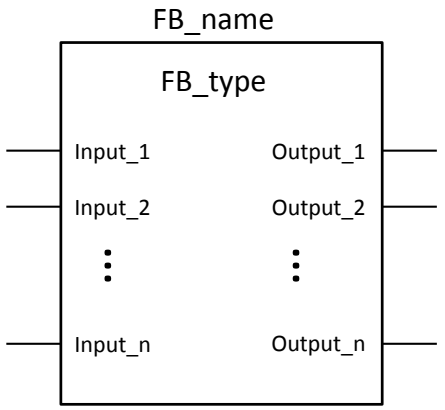
Program Organization Units: The function

- A function, as defined by IEC61131, has the following characteristic properties:
- It has one or more inputs (but no input is not allowed);
- It has exactly one output value (which may be a structure);
- A function has no `memory` from one call to the next, and it will return always the same output when given the same inputs.
- On every call to a function, all inputs have to be supplied.
- A function may use local variables for intermediate storage, but the value of these local variables will not be kept from one call to the next.
- A function may call other functions, but it is not allowed to call instances of function blocks.

Programing in function block diagram

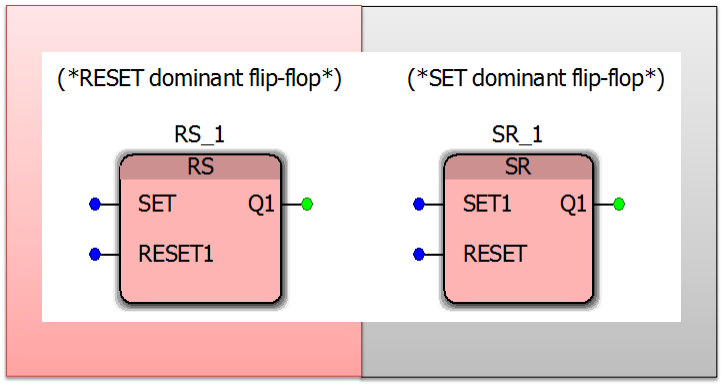
- Function block is a graphical programming language, which is akin to the electrical and block diagrams of the analog and digital technique.
- It mostly expresses combinatorial logic, but its blocks may have a memory (e.g. flip-flops).
- Basically, the language elements available in function block diagram are blocks and signals.
- The block is defined by its data flow interface (number and type of input/output signals) and shows Black-Box behavior.
- Function blocks are typed - the types of connection, input and output must match.
- Signals connect the function blocks. Connections carry a pseudo-continuous data flow.
- Each signal is connected to exactly one source. This source can be the output of a function block or a plant signal.
- The type of the output pin, the type of the input pin and the signal type must be identical.

Diagram representation of a Function Block

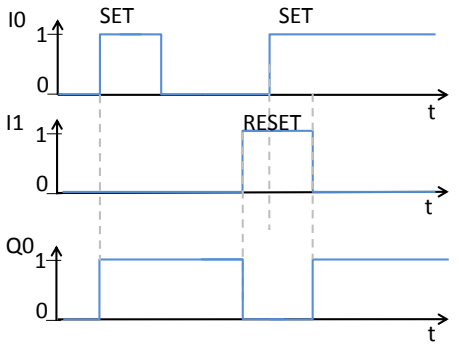


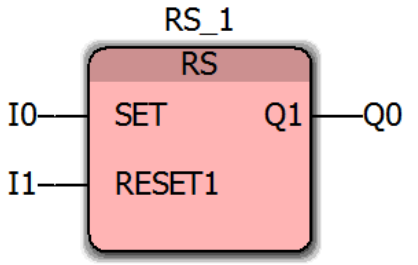
Function block type and the formal input(s)	Formal output(s)	Meanings
R/S flip-flop		
SR (S1,R)	Q	SET dominant
RS (R,S1)	Q	RESET dominant
Edge detections		
R_TRIG (CLK)	Q	rising edge
F_TRIG (CLK)	Q	falling edge
Counters		
CTU (CU,R,PV)	Q,CV	count-up
CTD (CD,LD,PV)	Q,CV	count-down
CTUD (CU,CD,R,LD,PV)	QU,QD,CV	count up-down
Timers		
TP (IN,PT)	Q,ET	Pulse
TON (IN,PT)	Q,ET	on-delay
TOF (IN,PT)	Q,ET	off-delay
RTC (EN,PDT)	Q,CDT	real-time clock

Flip-flops

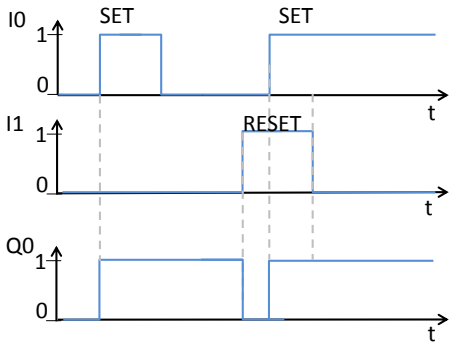


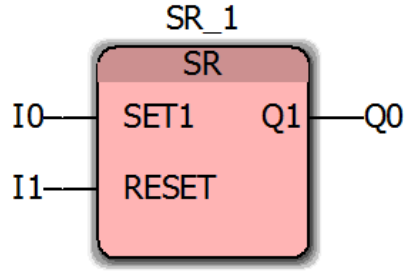
Time diagram of the RESET dominant flip-flop





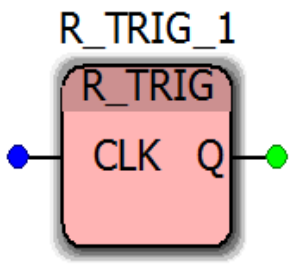
Time diagram of the SET dominant flip-flop



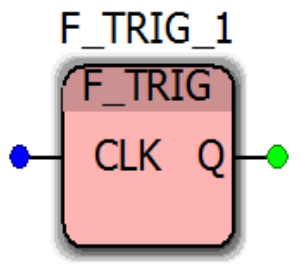


Edge detections

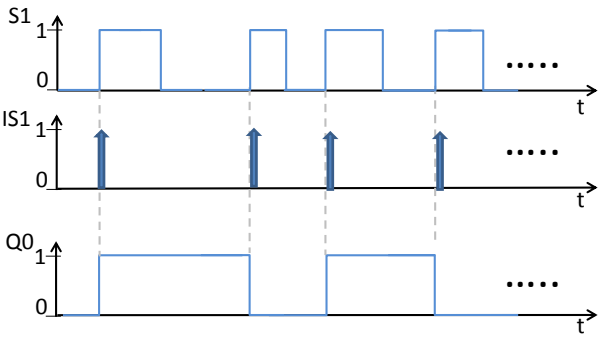
(*Raising edge *)



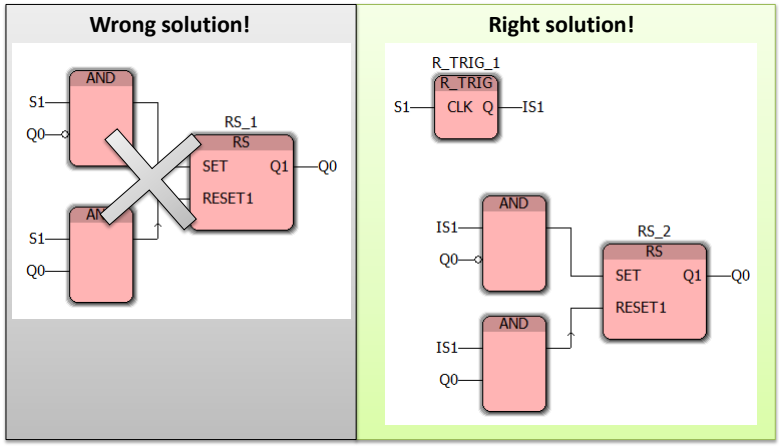
(*Falling edge *)



Example for using edge detections and flip-flops



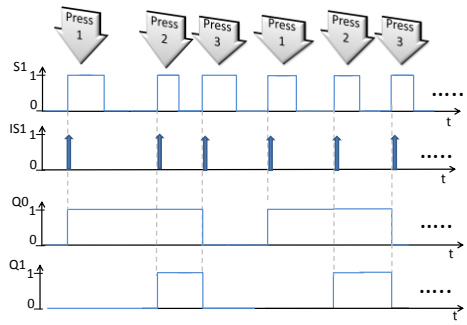
Why is the first solution wrong?



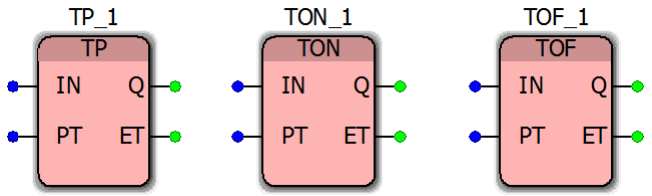
Exercise

Try to extend the flip-flop program above: you have one more lamp.

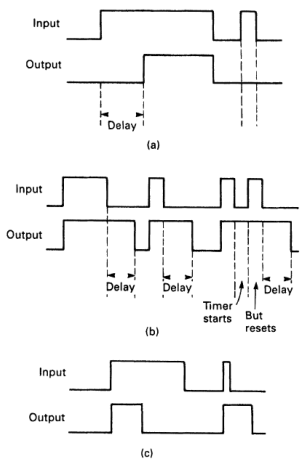
- The first pressure of the button switches ON the first lamp (Q0);
- The second pressure switches ON the second lamp (Q1), first lamp remains ON state;
- The third pressure of the button switches OFF both lamps;
- and so forth.



Timers

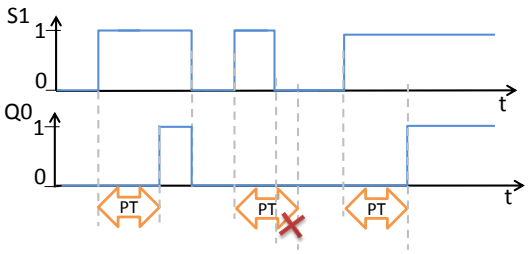


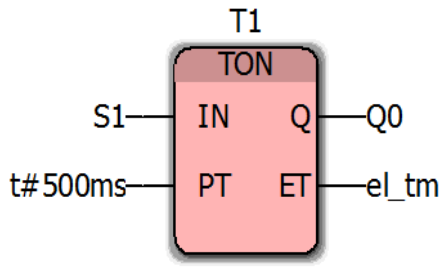
Timers



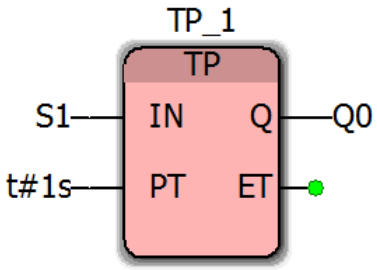
- *Different forms of timer:*
- *(a) on delay;*
- *(b) off delay;*
- *(c) fixed width pulse*

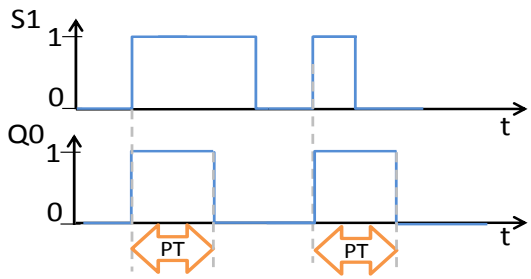
TON timer



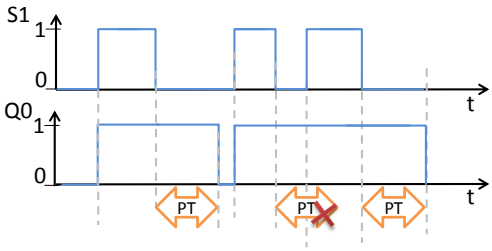


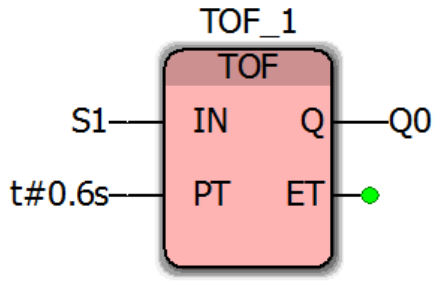
TP timer



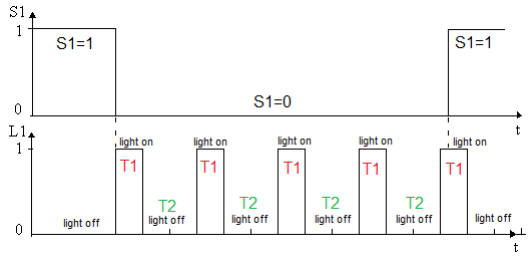


TOF timer





Flashing light example (alarm signal)

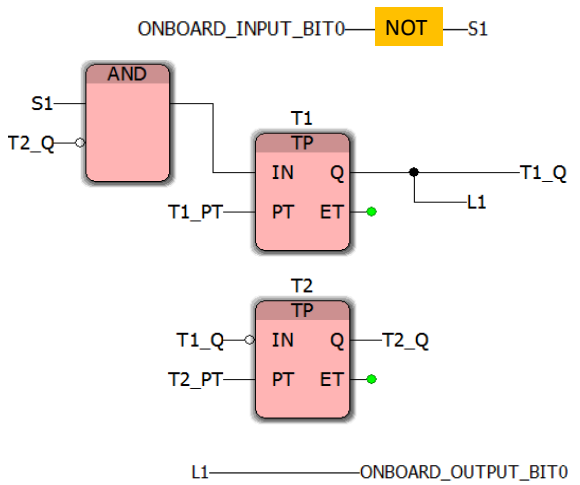


Inputs and Outputs

Inputs	Sign.	Logic	Address
Switch	S1	switched on: S1=1	Onboard_input_bit0
Outputs			
Lamp	L1	Light: L1=1	Onboard_output_bit0

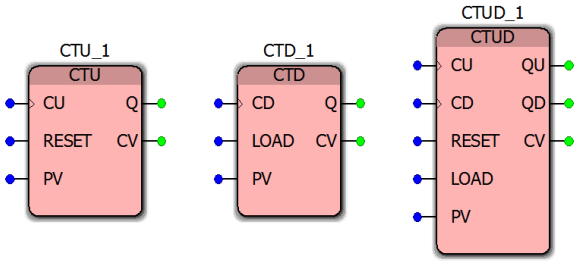
FBD

(*We can use our own variables: loading input into a variable S1*)

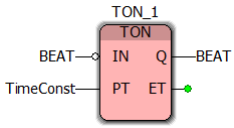


(*We can use our own variables: setting variable into output*)

Counters

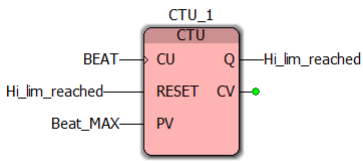


I/O	Meanings	Data type
R	RESET input	BOOL
S	SET input	BOOL
R1	RESET dominant	BOOL
S1	SET dominant	BOOL
Q	bool output	BOOL
CLK	ClocK	BOOL
CU	Count Up input	R_EDGE
CD	Count Down input	R_EDGE
LD	LoaD counter value	INT
PV	Preset Value counter	INT
QD	counter bool output (Down) QD=1, if CV=0	BOOL
QU	counter bool output (Up) QU=1, if CV≥PV	BOOL
CV	current counter value (Current Value)	INT
IN	timer start (INput)	BOOL
PT	timer Preset Time	TIME
ET	Elapsed Time	TIME
PDT	Preset Date and Time	DT
CDT	Current Date and Time	DT



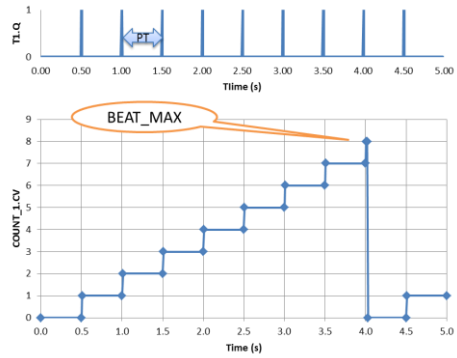
TimeConst : TIME := 0.5s;

Beat_MAX : INT :=8;

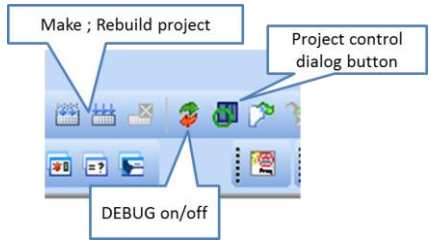


Application example: cycled, timed pulse series generation

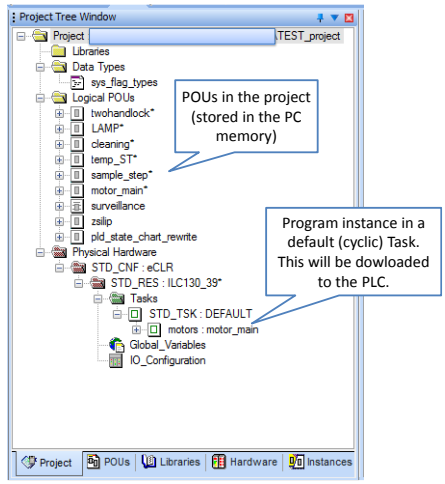
This timer and counter are the main components of function block based timed control solutions. (E.g. traffic light.)



PC WORX manages the downloading process by a separate project control dialog window.



Before compiling, don’t forget assign (instantiate) the appropriate program to the task



Intelligent control systems

SFC PROGRAMMING IN PCWORX SOFTWARE SYSTEM TRAFFIC LIGHTS PROGRAMMING GUIDE

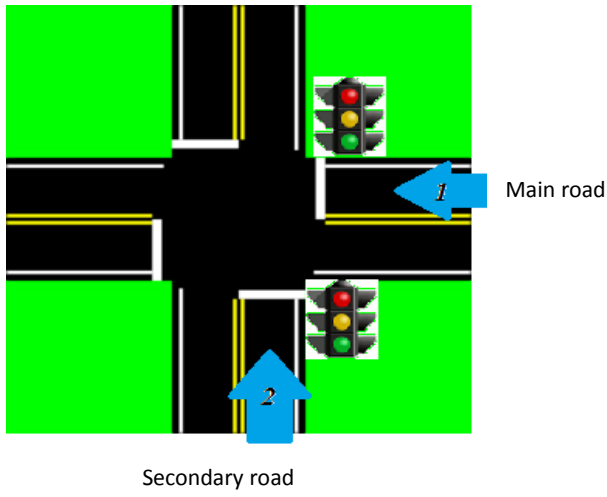
The *Sequential Function Chart,* SFC

- The *Sequential Function Chart* (SFC) programming method is used for a pictorial representation of a system's operation to show the sequence of the events involved in its operation.
- The operation is described by a number of separate sequentially connected states or steps that are represented by rectangular boxes, each representing a particular state of the system being controlled.
- Each connecting line between states has a horizontal bar representing the transition condition that has to be realized before the system can move from one state to the next.
- When the transfer conditions to the next state are realized, the next state or step in the program occurs. The process thus continues from one state to the next until the entire machine cycle is completed.
- Outputs/actions at any state are represented by horizontally linked boxes and occur when that state has been realized.

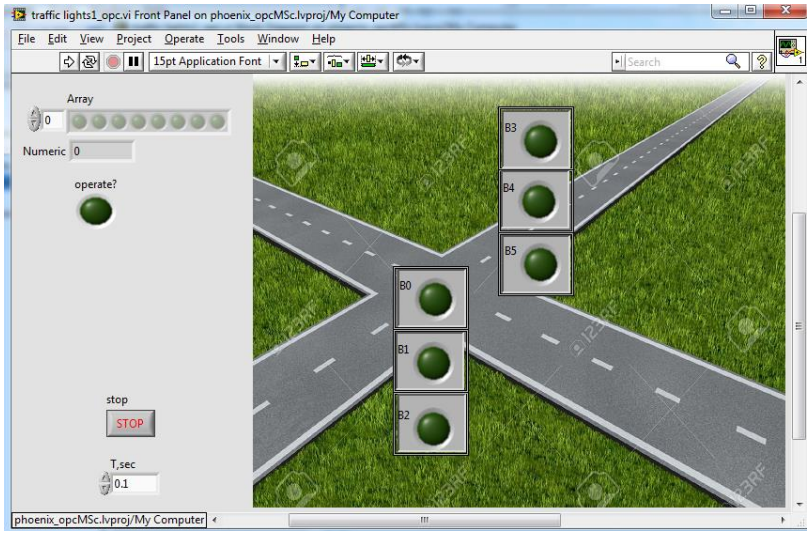
The *Sequential Function Chart,* SFC

- The sequential program consists of states connected by transitions.
- A state is activated by the presence of a token (the corresponding variable becomes TRUE).
- The token leaves the state when the transition condition (event) on the state output is true.
- Only one transition takes place at a time.
- Rule: there is always a transition between two states; there is always a state between two transitions.
- SFC programming implementations can be very different in various development systems.

Problem



Traffic light control display in LabVIEW



Inputs and Outputs

Inputs	Sign.	Logical assignment		address
ON/OFF switch	S0	ON:	S0=1	ONBOARD INPUT BIT0
Outputs				
RED light of main road	Red1	light on, when:	Red1=1	%Q5.0
YELOW light of main	Yellow1	light on, when:	Yellow1=1	%Q5.1
GREEN light of main	Green1	light on, when:	Green1=1	%Q5.2
RED light of secondary	Red2	light on, when:	Red2=1	%Q5.3
YELOW of secondary	Yellow2	light on, when:	Yellow2=1	%Q5.4
GREEN of secondary	Green2	light on, when:	Green2=1	%Q5.5

Operation requirements

If there is no operation, yellow lights are on.

After switching on, first both direction be red (2s)

Then red and yellow in direction 1 with red in direction 2 (2s)

After that begins the lamp cycle

The schedule chart of the lamp cycle

beat	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	0
R1										■	■	■	■	■	■	■	
Y1									■							■	
G1	■	■	■	■	■	■	■	■									■
R2	■	■	■	■	■	■	■	■	■	■						■	■
Y2										■					■		
G2											■	■	■	■			

Define the Steps

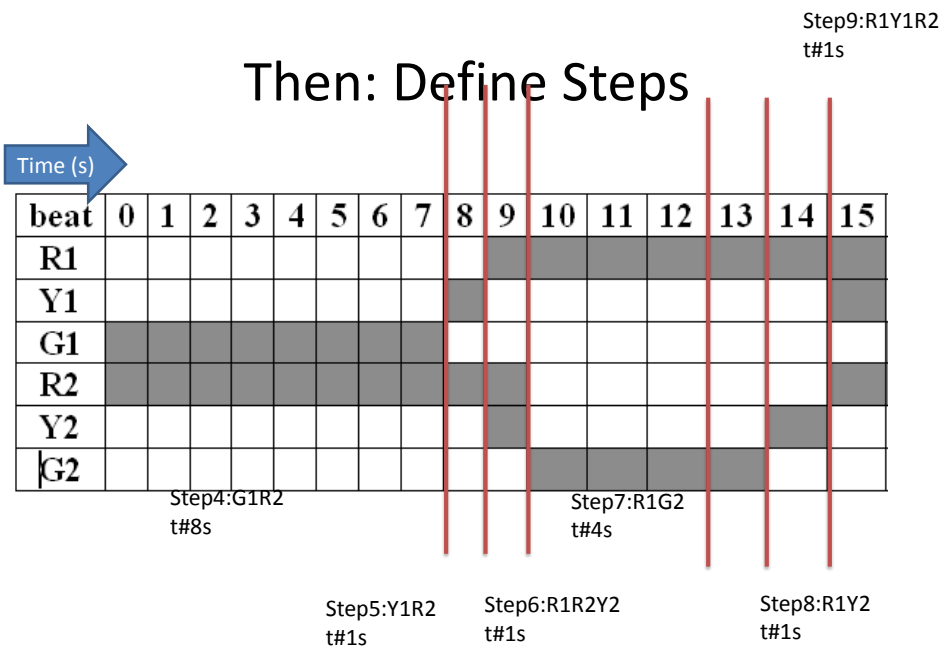
- Let one step be a constant lamp combination
- Transition happens, when the given lamp phase time elapsed
- Starting process:

  Step1:Y1Y2

  Step2 :R1R2

  Step3: R1Y1R2

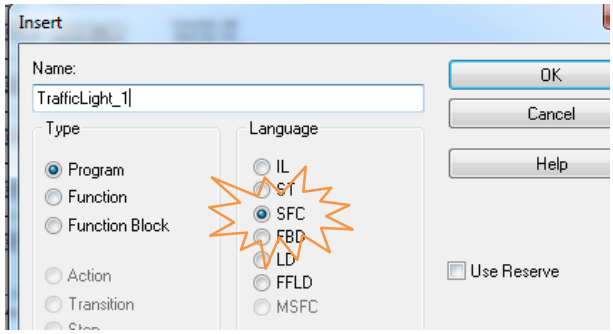
Then: Define Steps



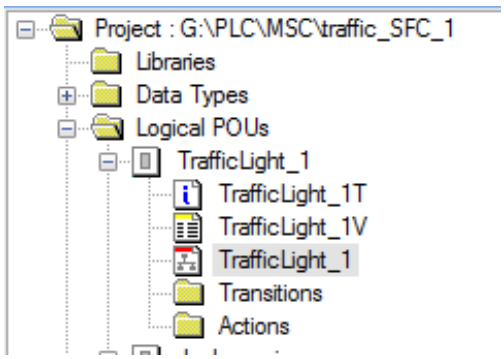
Solution in SFC

Open your project

On the project tree select Logical POUs

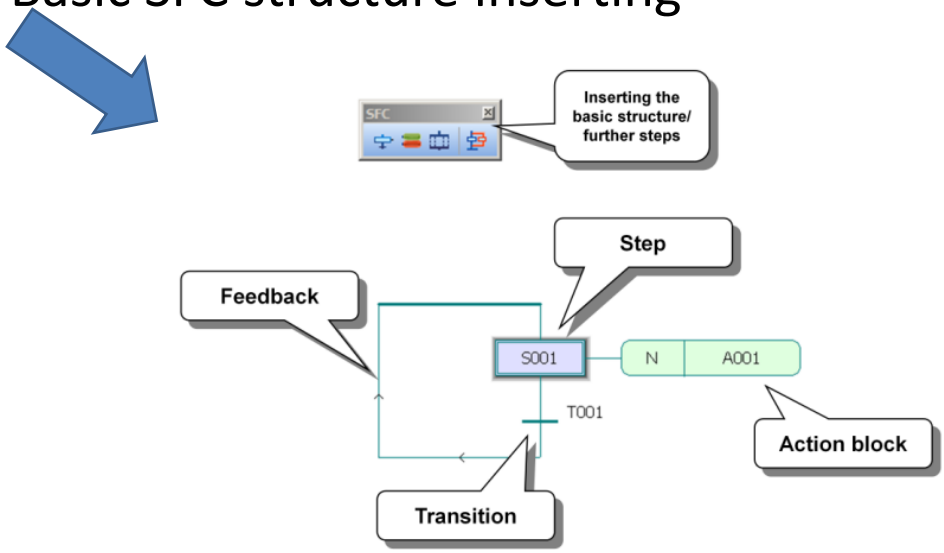


Open the program:

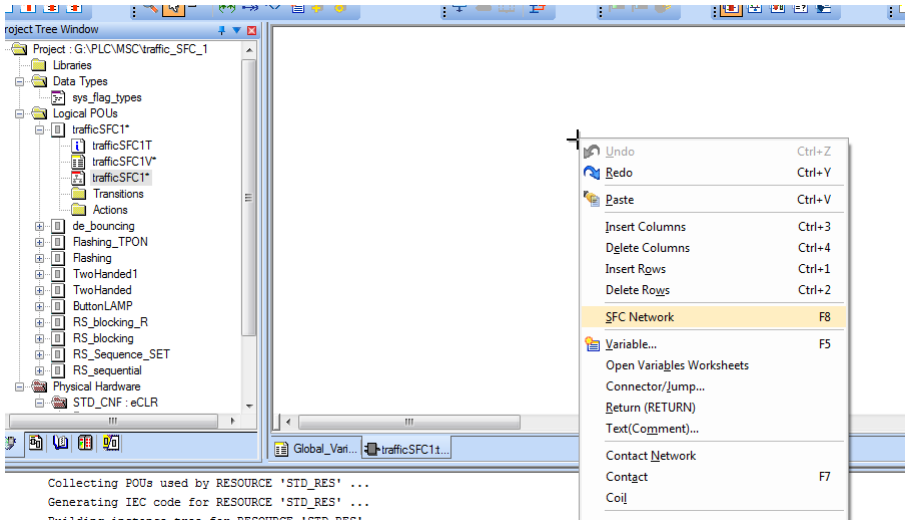


Insert SFC

Basic SFC structure inserting

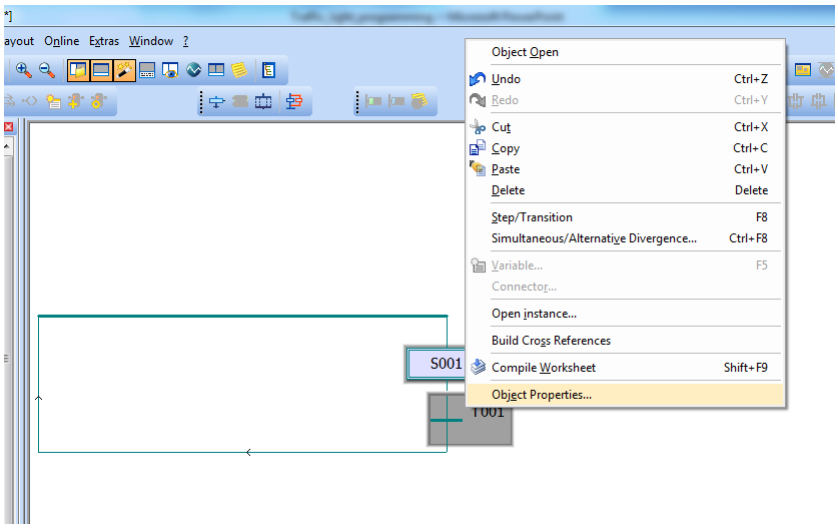


or

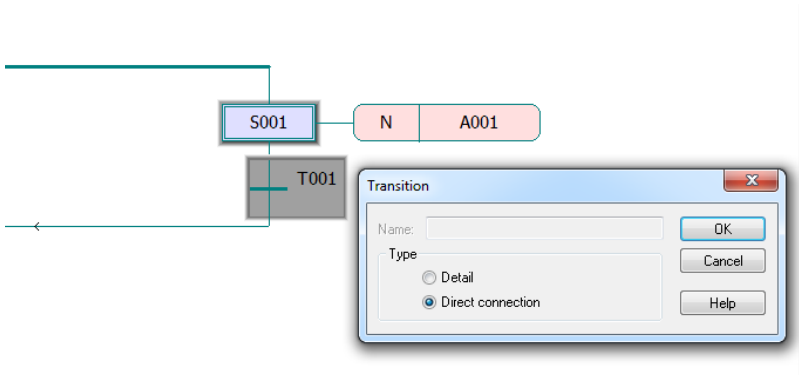


The initial SFC

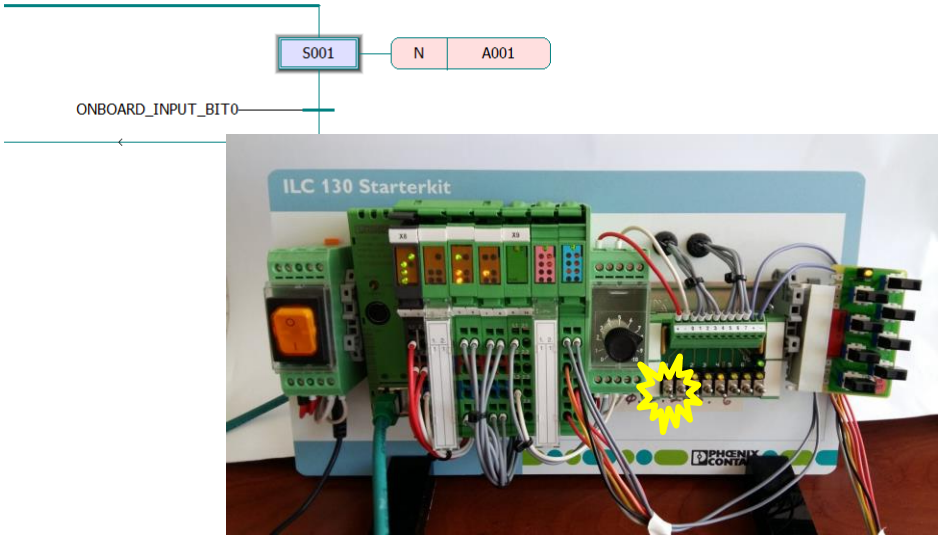
Change Transition properties…



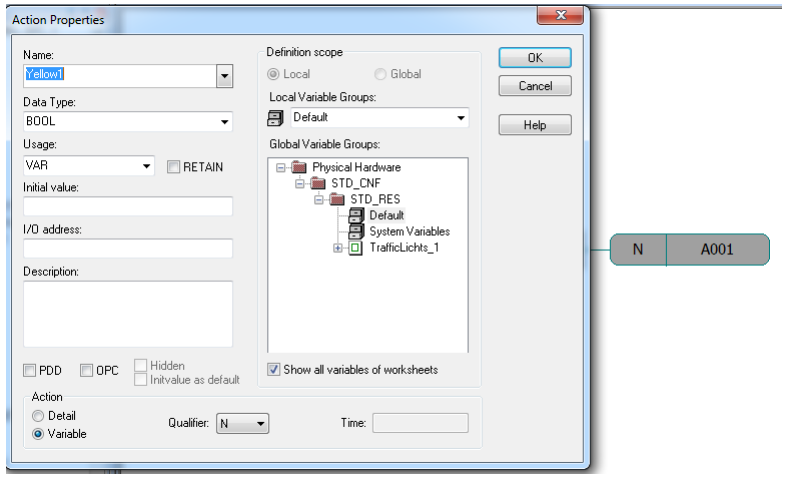
…to direct connection

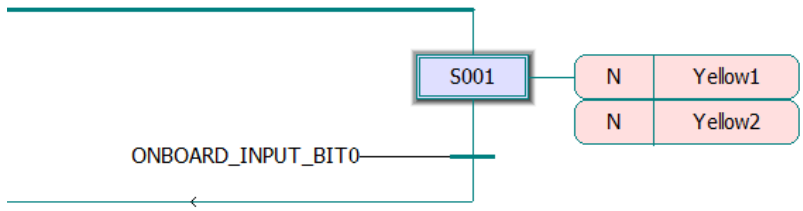


Let ONBOARD_Input_BIT0 be the Start switch



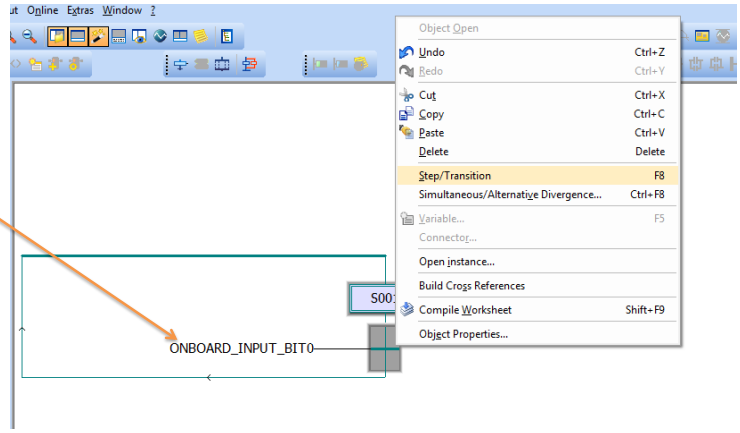
In the init step: yellow lamps be on





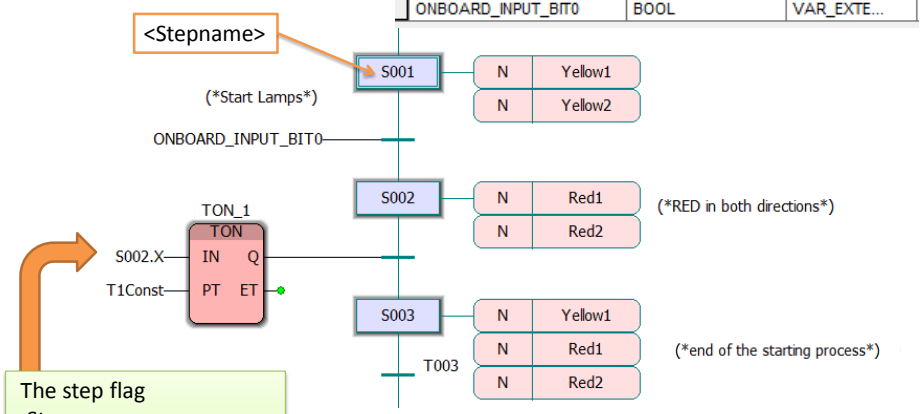
Add new step and transition

1. Select the transition
2. Right click
3. Select:
   Step/transition

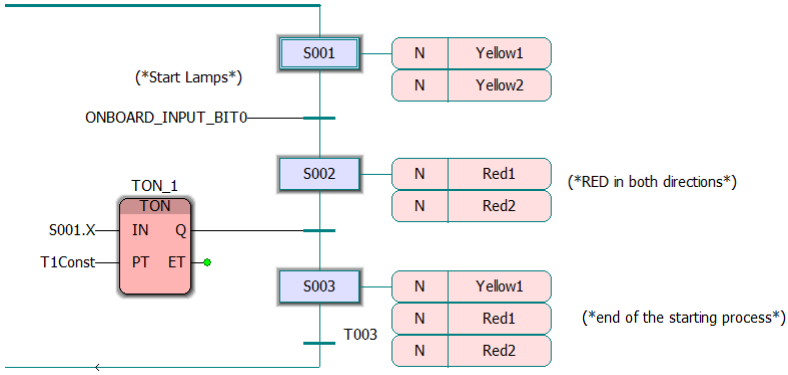


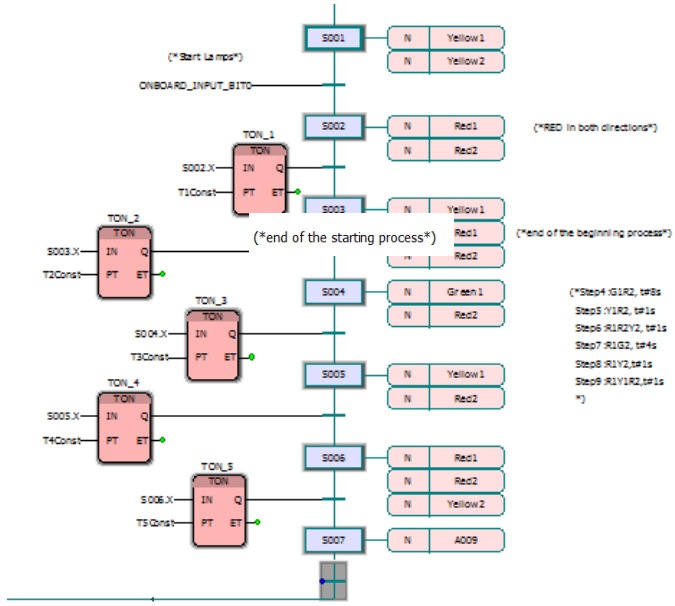
Declaration is automatic….but…

Default		
Yellow1	BOOL	VAR
Yellow2	BOOL	VAR
Red1	BOOL	VAR
Red2	BOOL	VAR
TON_1	TON	VAR
T1Const	TIME	VAR
ONBOARD_INPUT_BIT0	BOOL	VAR_EXTE...



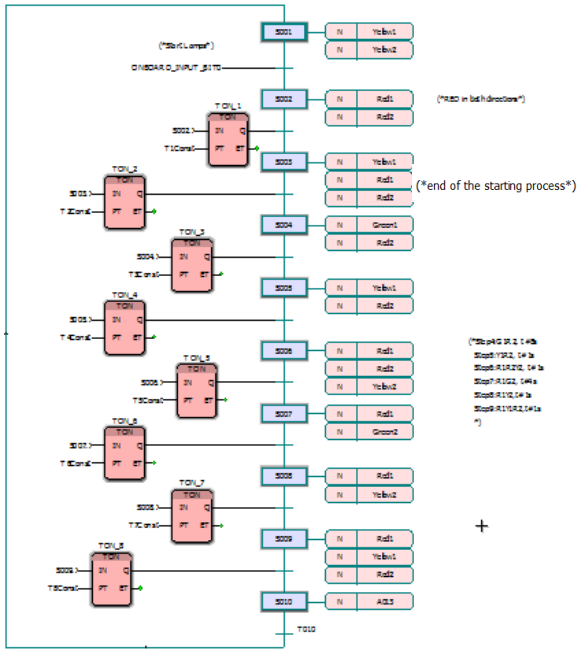
The step flag <Stepname>.x can be used without declaration. (System default variable.)


S001
N
Yellow1
N
Yellow2
(*Start Lamps*)
ONBOARD_INPUT_BIT0
S002
N
Red1
N
Red2
(*RED in both directions*)
TON_1
TON
S001.X
IN
Q
T1Const
PT
ET
S003
N
Yellow1
N
Red1
N
Red2
(*end of the starting process*)
T003

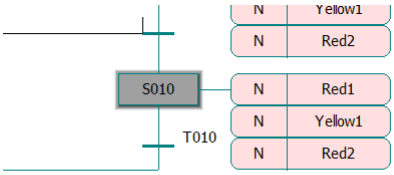

S001
N
Yellow1
N
Yellow2
(*Start Lamps*)
ONBOARD_INPUT_BIT0
S002
N
Red1
N
Red2
(*RED in both directions*)
TON_1
TON
S002.X
IN
Q
T1Const
PT
ET
S003
N
Yellow1
N
Red1
N
Red2
(*end of the beginning process*)
(*end of the starting process*)
TON_2
TON
S003.X
IN
Q
T2Const
PT
ET
S004
N
Green1
N
Red2
(*Step4:G1R2, t#8s
Step5:Y1R2, t#1s
Step6:R1R2Y2, t#1s
Step7:R1G2, t#4s
Step8:R1Y2,t#1s
Step9:R1Y1R2,t#1s
*)
TON_3
TON
S004.X
IN
Q
T3Const
PT
ET
S005
N
Yellow1
N
Red2
TON_4
TON
S005.X
IN
Q
T4Const
PT
ET
S006
N
Red1
N
Red2
N
Yellow2
TON_5
TON
S006.X
IN
Q
T5Const
PT
ET
S007
N
A009

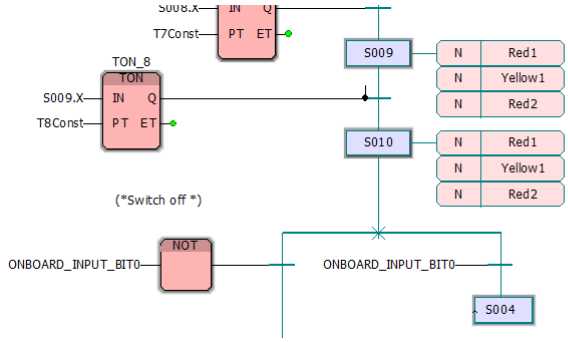
The lamp cycle is completed, then either starts again **OR** the operator switches off and the yellow lights will be on.

„OR" means alternative divergence

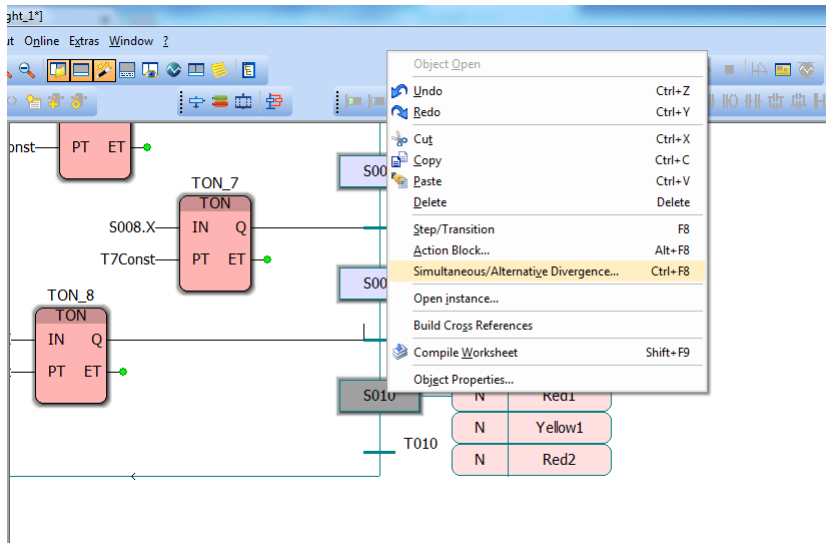


S10 is required only for inserting after it an alternative divergence

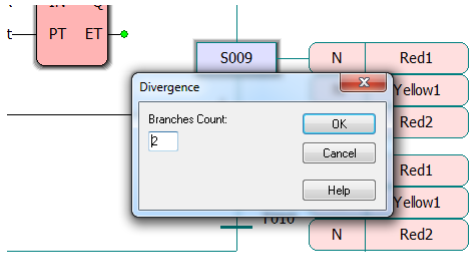




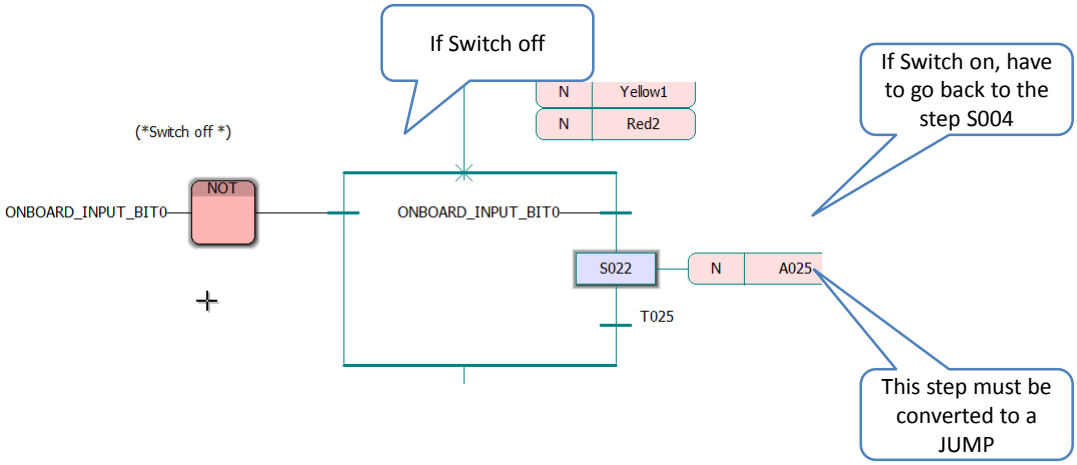
First select the step
then right-click and select :



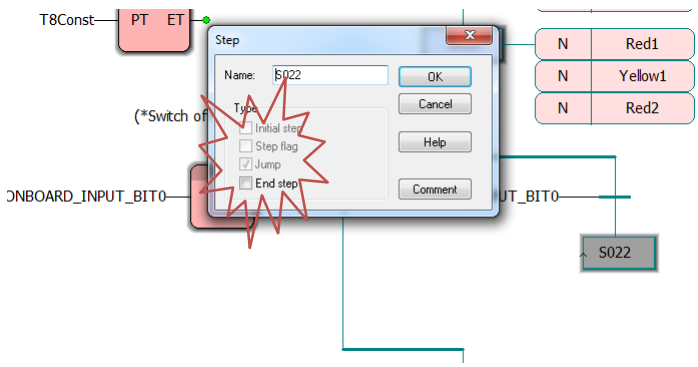
Select:



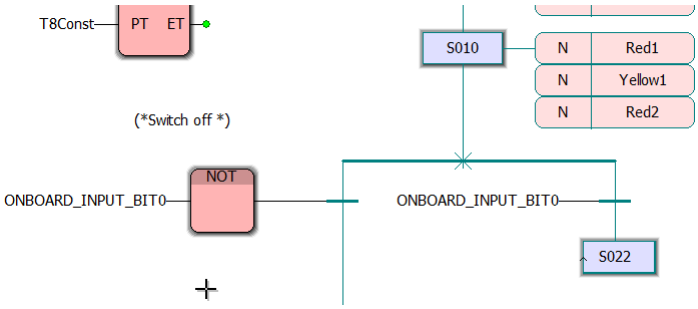
The result: „OR” branche



Duble click on the step and then mark JUMP:



The step number (S022) is wrong! Rename it!

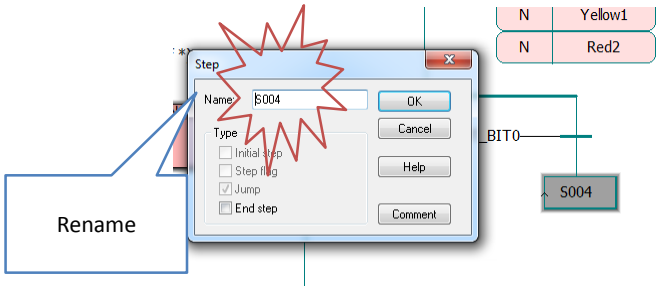


Jump

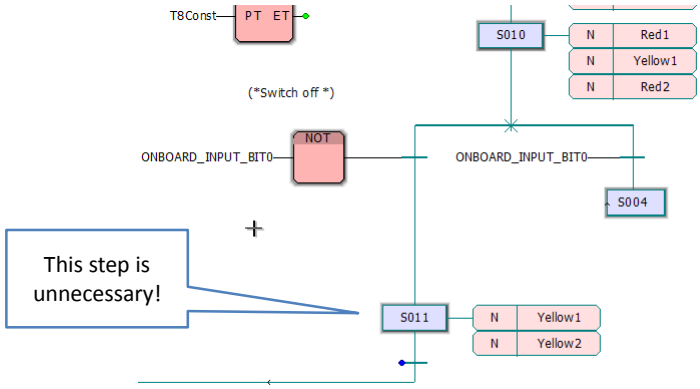
Unlike the other step types, a jump does not represent a process situation.

It has to be regarded as a direct jump to the target step indicated above the jump name.

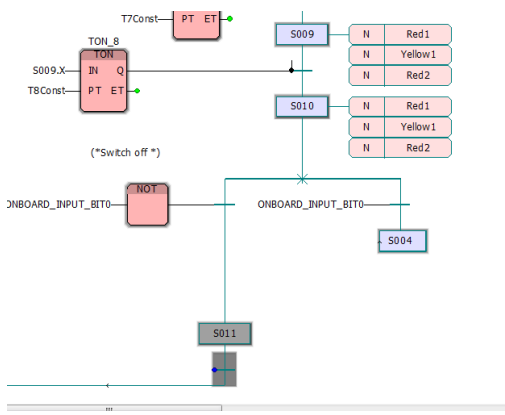
For the jump the exception holds true that it has to have the same name as another step.



End of SFC

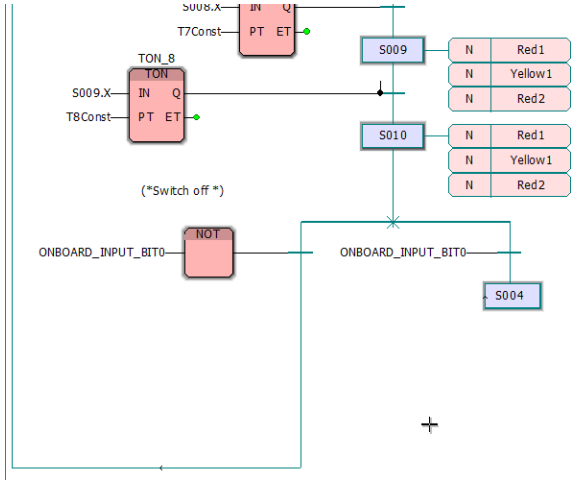


Select both of the step and transition

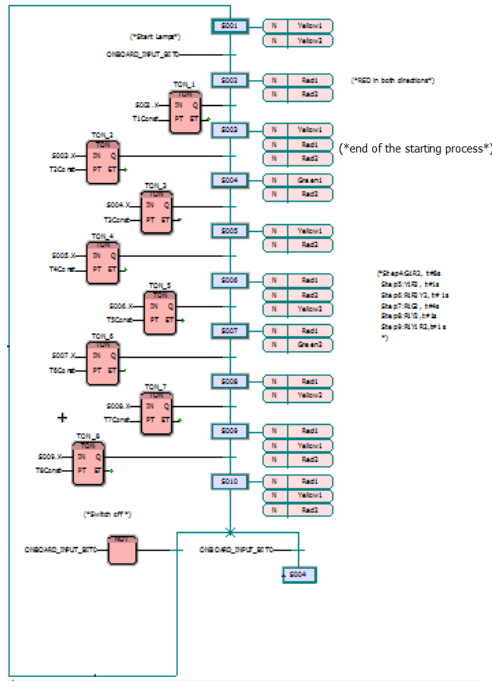


Then press the delete button !

The result:



The traffic lights SFC

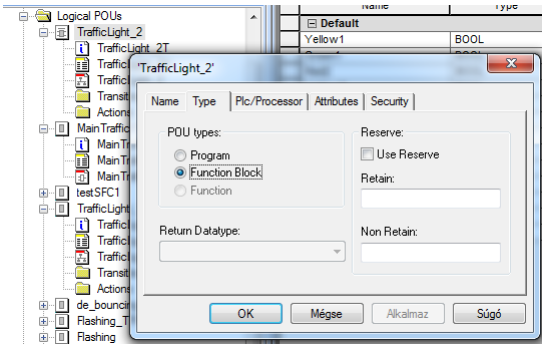


Converting program to sub-program: to a function block

Copy the program

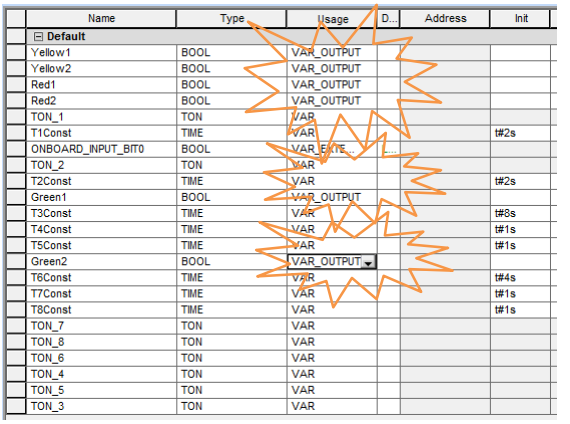
open the property window

Select type: Function Block



Assign the function block output variables in the declaration window

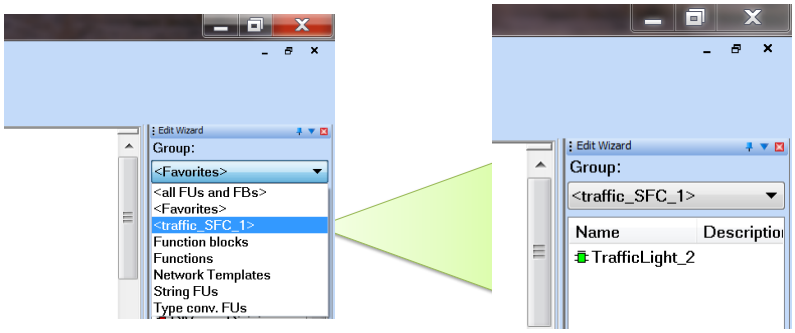
- No input is required, since ONBOARD INPUT BIT0 is global!



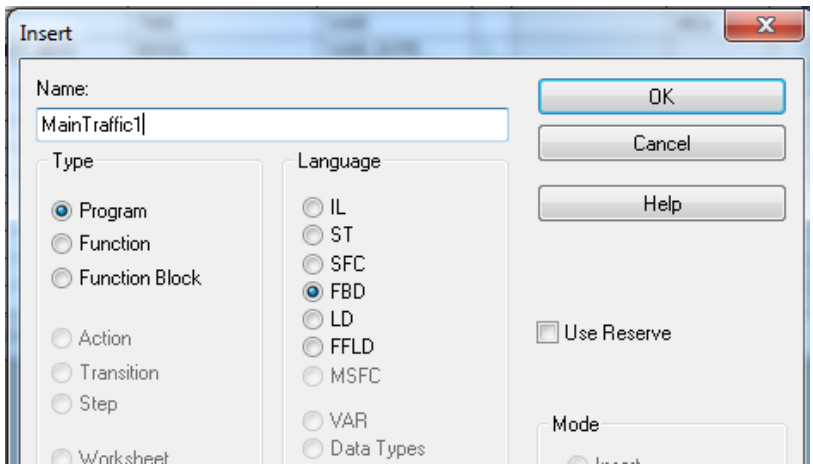
	Name	Type	Usage	D...	Address	Init
	Default					
	Yellow1	BOOL	VAR_OUTPUT			
	Yellow2	BOOL	VAR_OUTPUT			
	Red1	BOOL	VAR_OUTPUT			
	Red2	BOOL	VAR_OUTPUT			
	TON_1	TON	VAR			
	T1Const	TIME	VAR			t#2s
	ONBOARD_INPUT_BIT0	BOOL	VAR_EXTE...			
	TON_2	TON	VAR			
	T2Const	TIME	VAR			t#2s
	Green1	BOOL	VAR_OUTPUT			
	T3Const	TIME	VAR			t#8s
	T4Const	TIME	VAR			t#1s
	T5Const	TIME	VAR			t#1s
	Green2	BOOL	VAR_OUTPUT			
	T6Const	TIME	VAR			t#4s
	T7Const	TIME	VAR			t#1s
	T8Const	TIME	VAR			t#1s
	TON_7	TON	VAR			
	TON_8	TON	VAR			
	TON_6	TON	VAR			
	TON_4	TON	VAR			
	TON_5	TON	VAR			
	TON_3	TON	VAR			

Save all, then Rebuild the project !

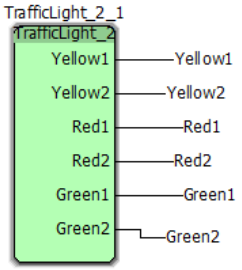
- Now you can see the FB in the right window

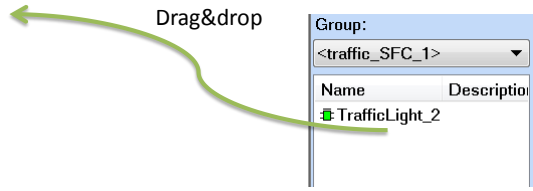


Insert a program, select language: FBD

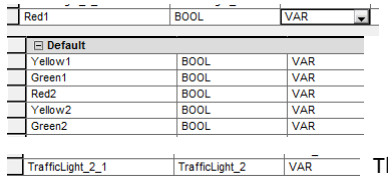


Drow the main program:





New variables, must be declared!



Red1	BOOL	VAR
Default		
Yellow1	BOOL	VAR
Green1	BOOL	VAR
Red2	BOOL	VAR
Yellow2	BOOL	VAR
Green2	BOOL	VAR
TrafficLight_2_1	TrafficLight_2	VAR

The FB is declared (automatic), type is the file name

Next step: Assinge the logical variables to a physical output

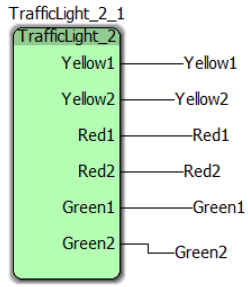
For example :Lamps are bools in QB5 byte

The declaration window:

Name	Type	Usage	Description	Address
Default				
Green1	BOOL	VAR		%Q5.2
Green2	BOOL	VAR		%Q5.5
Red1	BOOL	VAR		%Q5.0
Red2	BOOL	VAR		%Q5.3
TrafficLight_2_1	TrafficLigh...	VAR		
Yellow1	BOOL	VAR		%Q5.1
Yellow2	BOOL	VAR		%Q5.4

The main program

- Assign main traffic program to the resource default task (delet other program instances)
- Rebuild project
- If no error:
- Open Preject Control Dialog window
- Stop program in PLC if runs
- Download
- (igen…)
- Cold start
- Test the program operation



Intelligent control systems

OPC FOR TRAFFIC LIGHTS PROGRAMMING IN LABVIEW

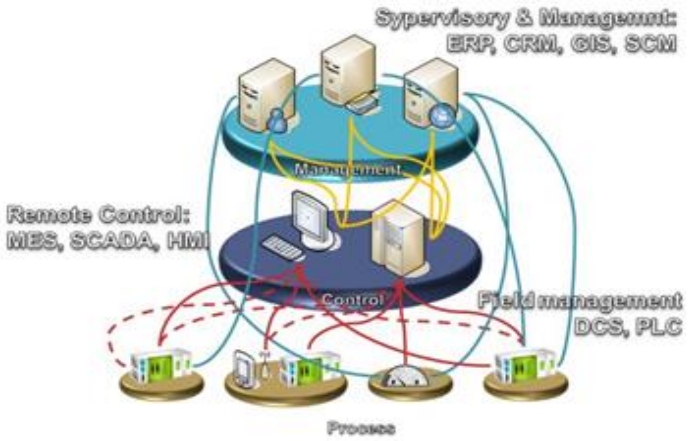
OPC references

- OPC: https://www.scribd.com/presentation/95424280/Ai-431-Opc-Common
- **Introduction to OPC:** http://www.ni.com/white-paper/7451/en/
  - **Connect LabVIEW to Any PLC Using OPC**

OPC

- (The acronym "OPC" comes from "OLE (Object Linking and Embedding) for Process Control".
- Since OLE is based on the Windows COM (Component Object Model) standard, OPC is essentially COM.
- Over a network, OPC relies on DCOM (Distributed COM)

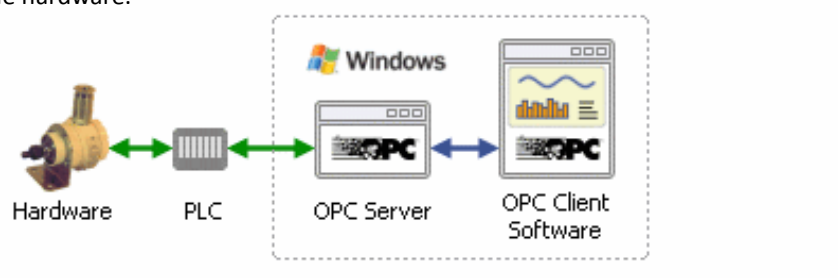
Control systems have hierarchical structure

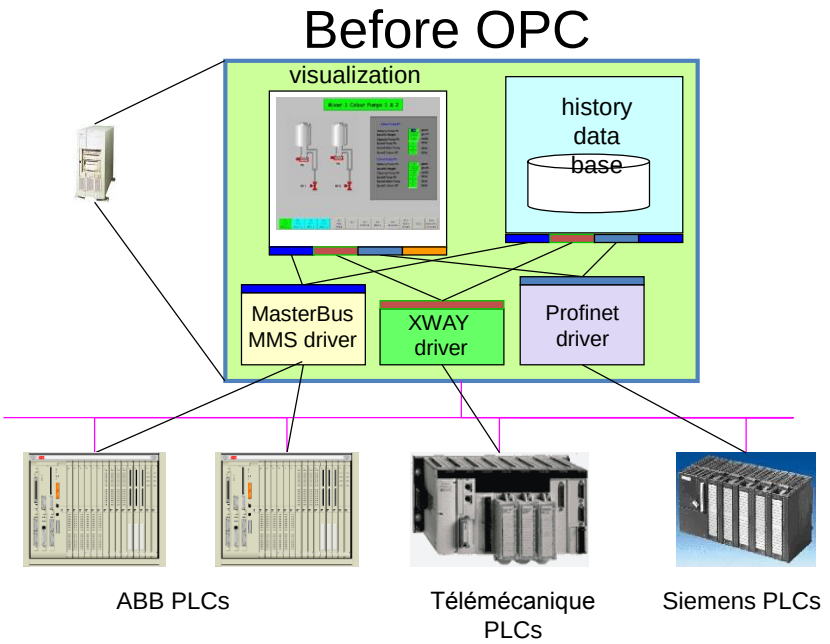


- A factory automation system or process consists of different controllers and devices from different suppliers or vendors with different protocols.
- These controllers and devices are essential to communicate with business or management systems.
- OPC creates an environment to access real-time plant data from such vendors.

Classic OPC = OLE for Process Control

- OPC is a software interface standard that allows Windows programs to communicate with industrial hardware devices.
- OPC is implemented in server/client pairs.
- The OPC server is a software program that converts the hardware communication protocol used by a PLC into the OPC protocol.
- The OPC client software is any program that needs to connect to the hardware, such as an HMI .
- The OPC client uses the OPC server to get data from or send commands to the hardware.





OPC example: SCADA connection

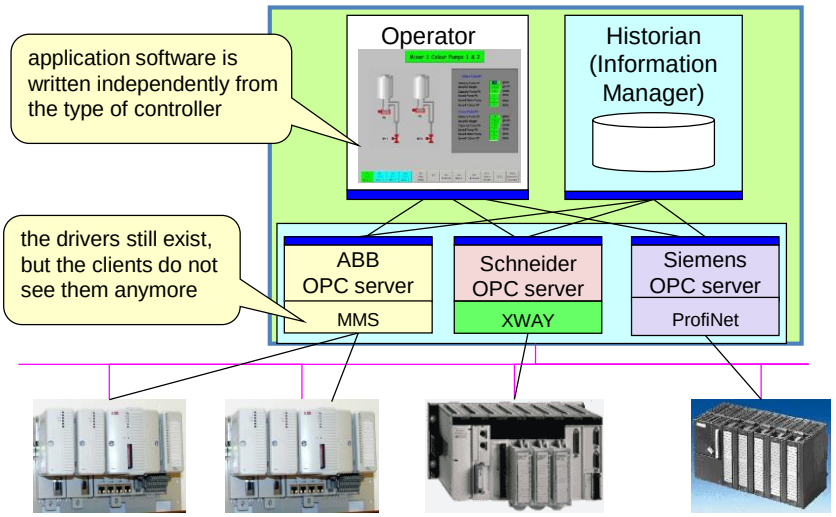


ABB AC800M Télémécanique TSX Siemens S7

Importance

OPC is the greatest improvement in automation since IEC 61131.

OPC is supported by the OPC foundation (http://www.opcfoundation.org/)

More than 150 vendors offer OPC servers to connect their PLCs, field bus devices,
displays and visualization systems.

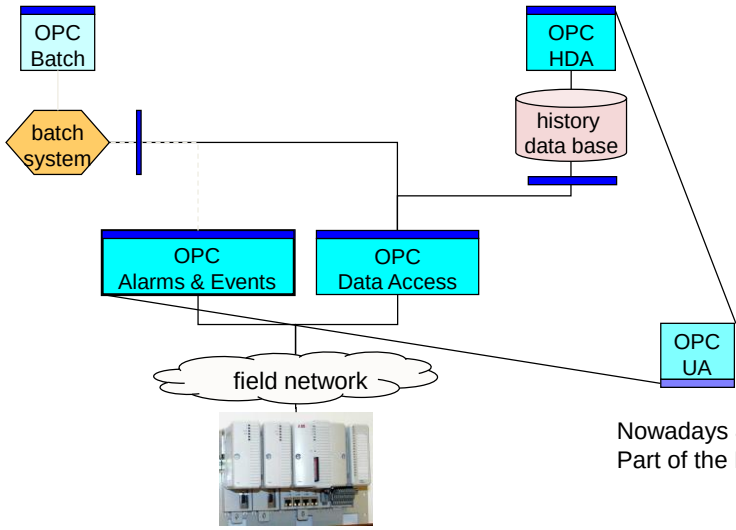
OPC is also used for data exchange between applications and for accessing databases

OPC is available as DLL for Automation Interface (Visual Basic,..) and Custom (C++,..)

Classic OPC consists of three major components:

1) OPC - DA = Data Access (widespread, mature)
2) OPC - AE = Alarms and Events (not yet much used)
3) OPC - HDA = Historical Data Access (seldom used)

The main OPC Specifications



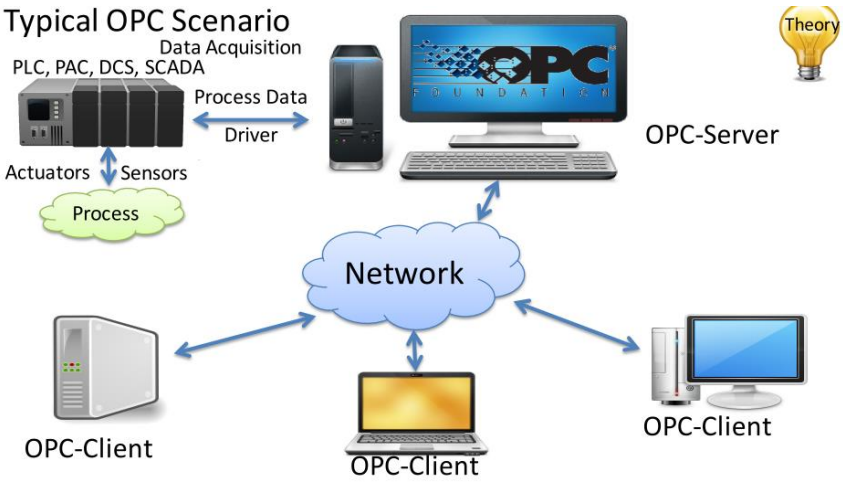
Nowadays & future
Part of the IoT

Beyond Microsoft: OPC UA

In a move to get more independence from Microsoft and use web technology, a new specification called " Unified Architecture" that uses web services for all kinds of transactions: query, read, write, subscribe,...

The classical OPC DA, AE and HDA are implemented with XML / SOAP / WSDL
this allows encryption and authentication of process data.

OPC UA does not only standardize the interfaces, but also the transmitted data.
Today the acronym OPC stands for Open Platform Communications or Open Process Control.



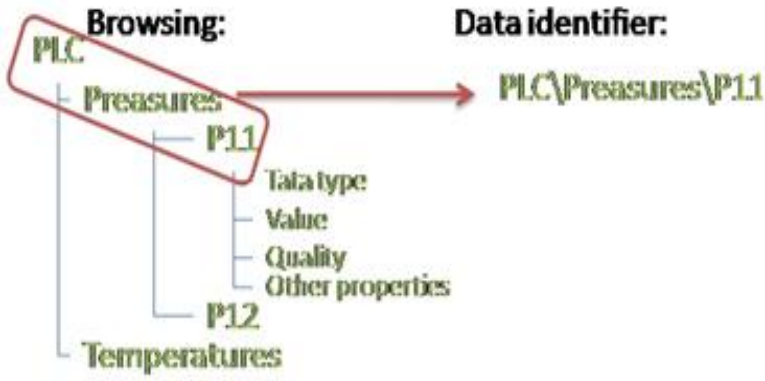
OPC Implementation

- OPC allows client and server applications to communicate with each other.
- OPC is designed to be an abstraction layer between industrial networks and proprietary PLC drivers.
- The OPC standard specifies the behavior that the interfaces are expected to provide to their clients; and the clients receive the data from the interfaces using standard function calls and methods.
- Consequently, as long as a computer analysis or data acquisition program contains an OPC client protocol, and an industrial device driver has an associated OPC interface, the program can communicate with the device.
- The specification also includes a protocol for working with data control systems and application databases, as well as online data access, alarm and event handling, and historical data access for all of these data sources.

The data access server has three divisions

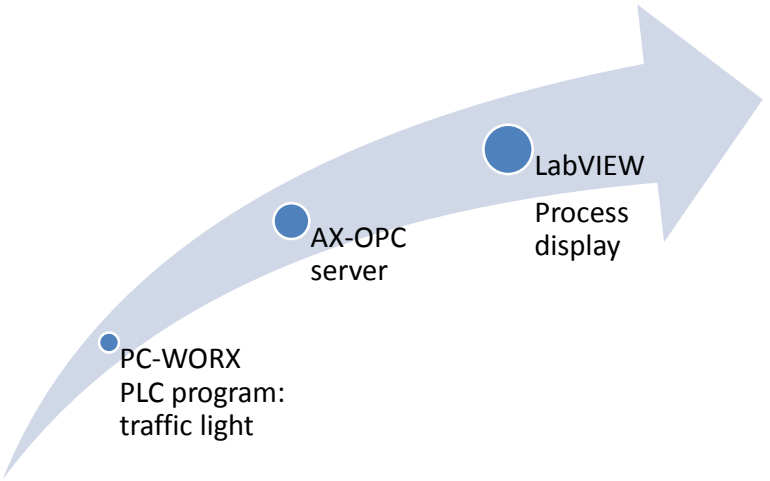
- **Server** – Contains all of the group objects
- **Group** – Maintains information about itself and contains and organizes the OPC items
- **Item** – Contains a unique identifier held within the group. *The identifier acts as a reference for the individual data source*:
  - value, quality, and timestamp information. (TAG)
- The value is the data from the source. The quality status gives information about the device. The timestamp is the time that the data was retrieved.

Tag name and identifier



The basic OPC term used in the process of data publishing is a tag. It is a minimum data portion, which is addressable, i.e. has a unique identifier and may be indicated in a read/write operation.

Practice: programs in three levels

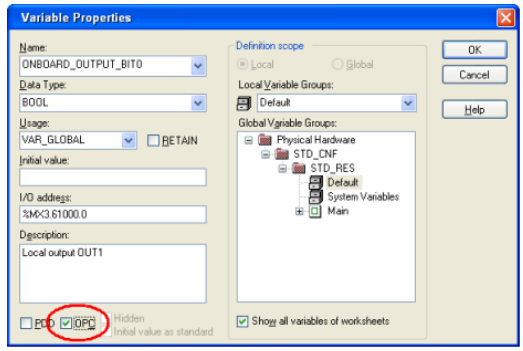


Semi-finished programs

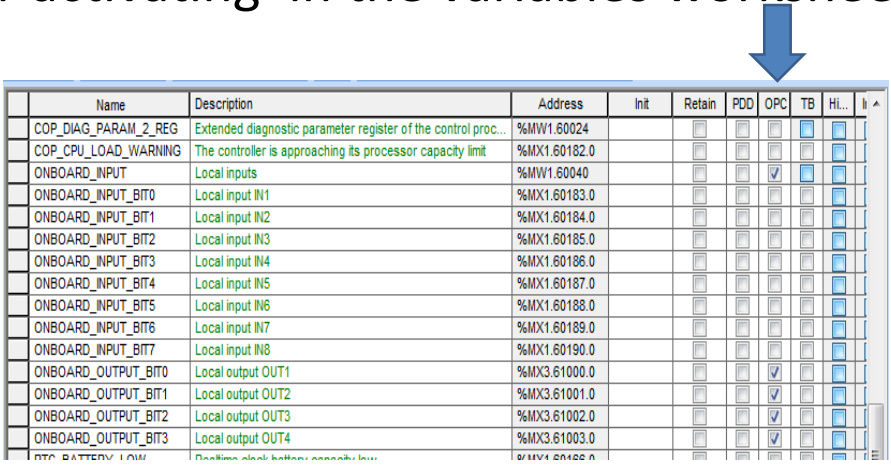
- Download from NEPTUN into your own directory then unzip:
  - PhoenixOpcLVMSc.zip
- This is a LabVIEW project we will use.

Creating OPC variables in PC Worx

- In order to use a variable in AX OPC SERVER, activate the "OPC" checkbox:
  – When creating variables in the "Variable Properties" window



Or activating in the variables worksheet



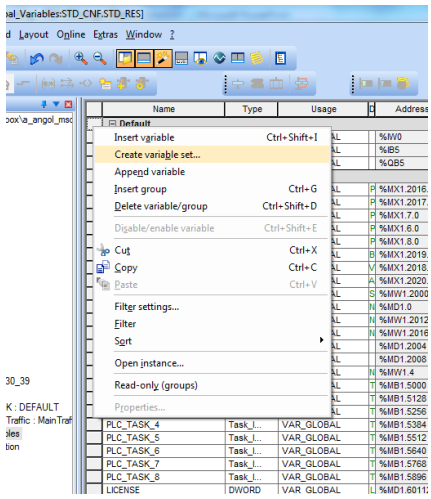
Name	Description	Address	Init	Retain	PDD	OPC	TB	Hi...
COP_DIAG_PARAM_2_REG	Extended diagnostic parameter register of the control proc...	%MW1.60024						
COP_CPU_LOAD_WARNING	The controller is approaching its processor capacity limit	%MX1.60182.0						
ONBOARD_INPUT	Local inputs	%MW1.60040				✓		
ONBOARD_INPUT_BIT0	Local input IN1	%MX1.60183.0						
ONBOARD_INPUT_BIT1	Local input IN2	%MX1.60184.0						
ONBOARD_INPUT_BIT2	Local input IN3	%MX1.60185.0						
ONBOARD_INPUT_BIT3	Local input IN4	%MX1.60186.0						
ONBOARD_INPUT_BIT4	Local input IN5	%MX1.60187.0						
ONBOARD_INPUT_BIT5	Local input IN6	%MX1.60188.0						
ONBOARD_INPUT_BIT6	Local input IN7	%MX1.60189.0						
ONBOARD_INPUT_BIT7	Local input IN8	%MX1.60190.0						
ONBOARD_OUTPUT_BIT0	Local output OUT1	%MX3.61000.0				✓		
ONBOARD_OUTPUT_BIT1	Local output OUT2	%MX3.61001.0				✓		
ONBOARD_OUTPUT_BIT2	Local output OUT3	%MX3.61002.0				✓		
ONBOARD_OUTPUT_BIT3	Local output OUT4	%MX3.61003.0				✓		

OPC data is also transmitted when the project is sent to the control system.

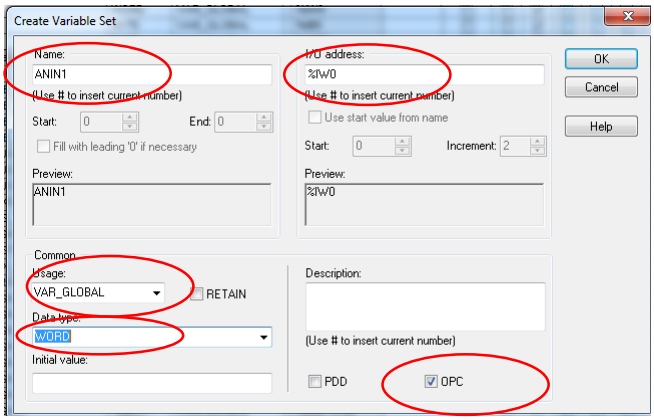
Comments

- the ONBOARD INPUT in the table above is only checked. This means that all onboard input bits are transferred to the OPC server database in a U16 word.
- the values of the input switches are on the lower eight bits of this word.
- The word query requires less telegraphy from the system, as if the inputs are bit-tagged. (Bits are also transmitted through byte or word packet on the communication channel).
- We also handle the binary signals of the expansion modules in bytes (IB5: black switches, QB5 pink leds).
- ANIN1 is the declaration of the analog input signal, the value can be changed by the black potentiometer.

Create new global variables



Declaration of the analog input variable



Declare all variables with physical addresses to global variables

- For each variable, tick the OPC rubric!

Name	Type	Usage	D	Address	Init	Retain	PDD	OPC
Default								
ANIN1	WORD	VAR_GLOBAL		%IW0		☐	☐	☑
IB5	BYTE	VAR_GLOBAL		%IB5		☐	☐	☑
QB5	BYTE	VAR_GLOBAL		%QB5		☐	☐	☑

Digital Output modul:
8bits in 1byte
(pink leds)

Digital Input modul:
8bits in 1byte
(black swithes)

Before testing

- Precondition:
  - your traffic light program is tested and is faultless
  - OPC variables are ticked in global variables
- Rebuild the project
- Download your project into the PLC
- By „ColdStart” run the traffic light program
- Close the „Project Control Dialog” window
- *The program runs in PLC!*

AX OPC SERVER

- The AX OPC SERVER is responsible for exchanging data between Windows applications and PC Worx -based controllers (PLC) from Phoenix Contact via the client/server interface OPC (OLE for Process Control).
- The communication connections between AX OPC SERVER and PLC are realized via TCP/IP protocol.
- The AX OPC SERVER supports the interface standards OPC DA.
- The AX OPC SERVER supports an automatic remote configuration by uploading the configuration data from the target device (PLC) during startup.
- When a project is modified in the running PLC process, the AX OPC SERVER will be updated automatically which ensures uninterrupted data exchange with the OPC client.

The AX OPC SERVER includes

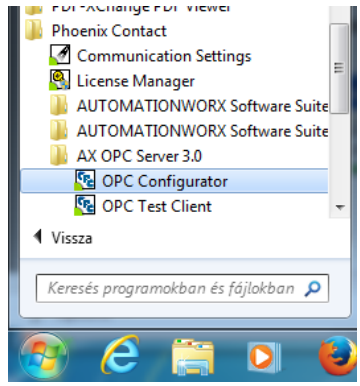
- The "OPC Configurator" for establishing a connection between PC Worx/PC Worx Express and an OPC client.
- An "OPC Test Client" for testing the connection.

- *The AX OPC SERVER is configured on all PC in the lab, no need to change except IP address.*

Checking the OPC Configurator

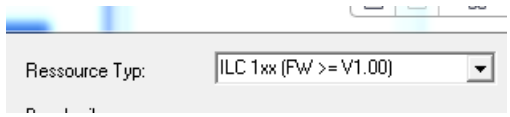
Start the "OPC Configurator" program:

in ALL PROGRAMS:



OPC onfigurator

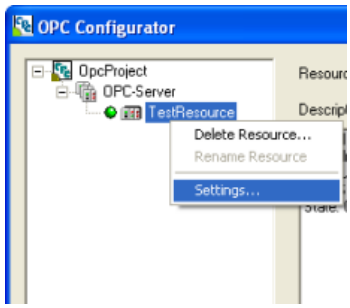
- The configurator commands can be accessed via context menus. Do’nt change the name of the resource .
- • The resource type :



- Check the IP address: if it differs from the IP address of the PLC wich you use, change the IP ADDRESS

Change the IP ADDRESS

Open the context menu for the "TestResource" entry.
• Select the "Settings..." menu item.



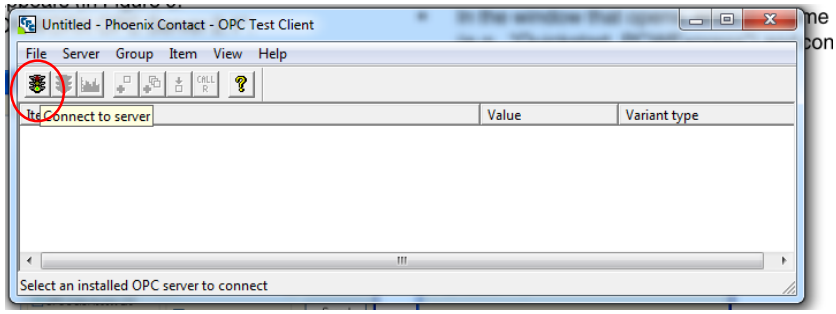
Under "IP Address" enter the IP address for the PLC for which you would like to use the OPC data.

Colse the configurator.

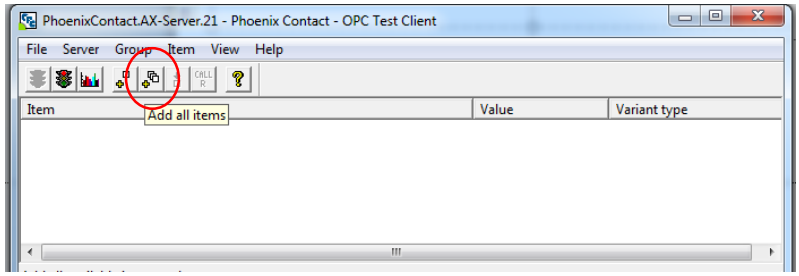
OPC Test Client

- Test Client can be used to test the OPC configuration.
- Open the "OPC Test Client" program.

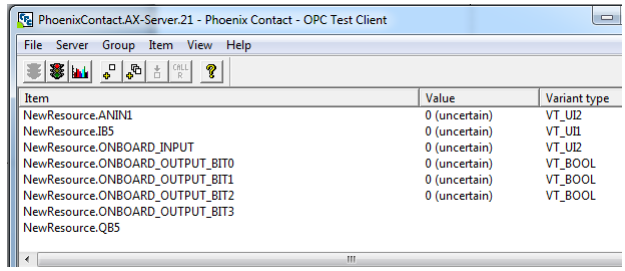
Press for connection



Select the "Add all Items" menu item

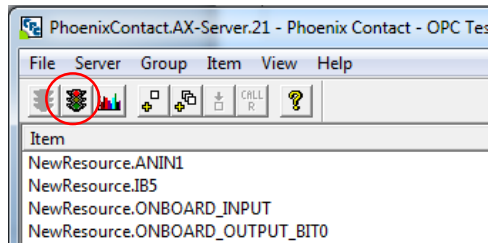


The connection between PC Worx, OPC, and Test Client can now be tested

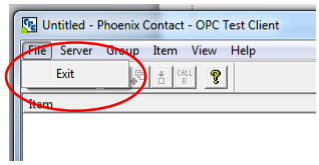


Closing the OPC test client

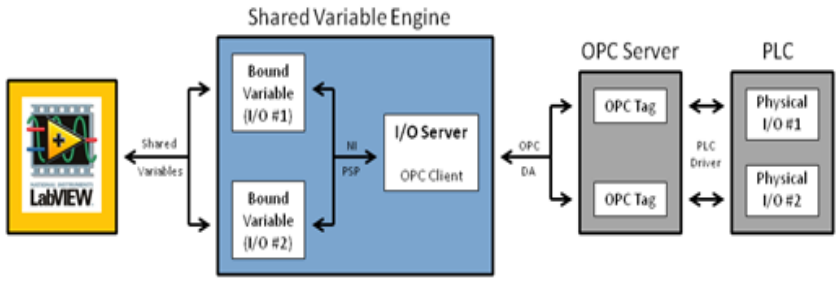
- Disconnect from server



- Then Close OPC test client program



Using LabVIEW as an OPC Client



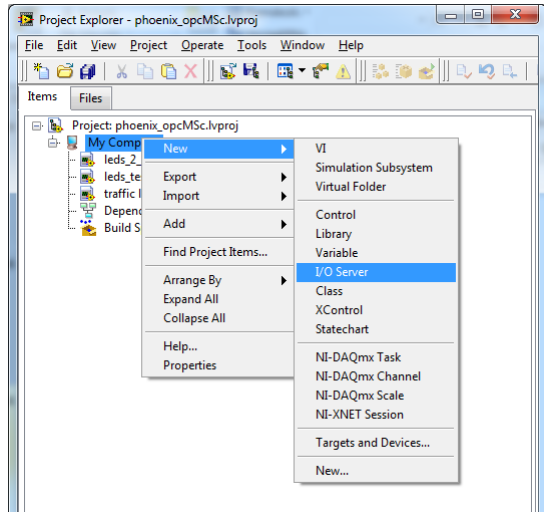
OPC Client I/O server will list all available OPC servers that are installed and running on a local or network computer

The OPC Client I/O servers can connect to each OPC tag using the OPC DA standard.

PLCs publish data to the network. An OPC Server program uses the PLC's proprietary driver to create OPC tags for each physical I/O on the PLC.

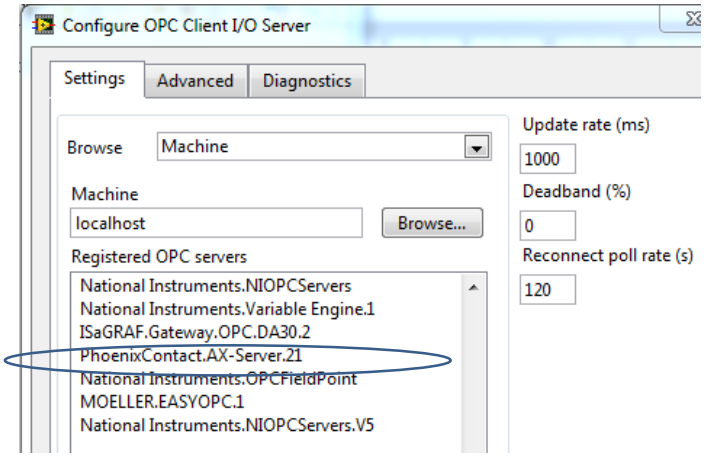
Connect LabVIEW to OPC Tags by Creating an I/O Server

- Open the LV project downloaded:
  PhoenixOpcLVMSc\phoenix_opcMSc.lvproj



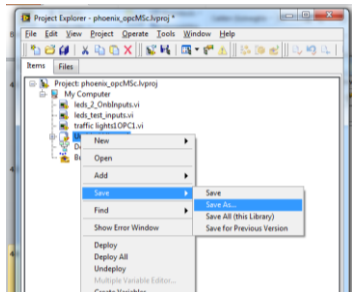
- In the LabVIEW Project window, right-click **My Computer** and select **New»I/O Server**

- Select **OPC Client** in the Create New I/O Server Window and click **Continue.**
- Choose PhoenixContact.AX-Server.21



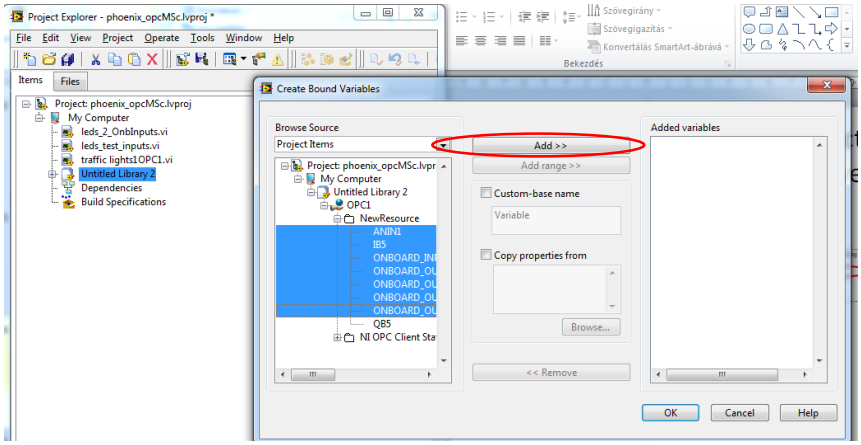
This creates a connection from LabVIEW to the OPC tags, which updates every 1000 ms.

- Select **OK**. A library is automatically created in your project explorer window to manage the I/O Server.
- You can rename the libary, for example phoenix.lvlib
- Save the project.



Create Shared Variables that Connect to the OPC Tags through the I/O Server

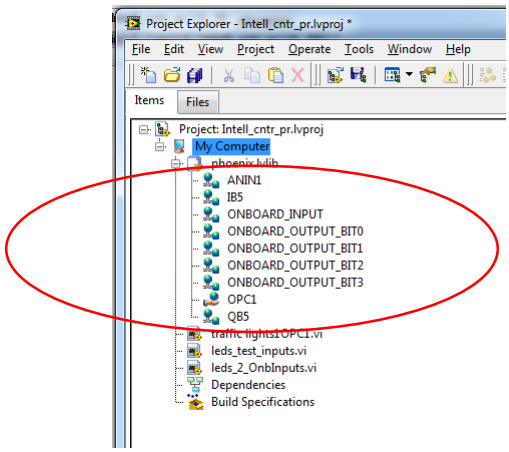
- Right-click the newly created library and select **Create Bound Variables…**



Create Shared Variables that Connect to the OPC Tags through the I/O Server

- Select all the items and click **Add** and **OK.** This creates shared variables that are bound to the PLCs' OPC tags and loads them into the Multiple Variable Editor.
- In the Multiple Variable Editor, select **Done**. This adds the new shared variables to the library that was created earlier.
- You now have access to PLC data natively in LabVIEW through the shared variables

Shared (Bound) variables in the project



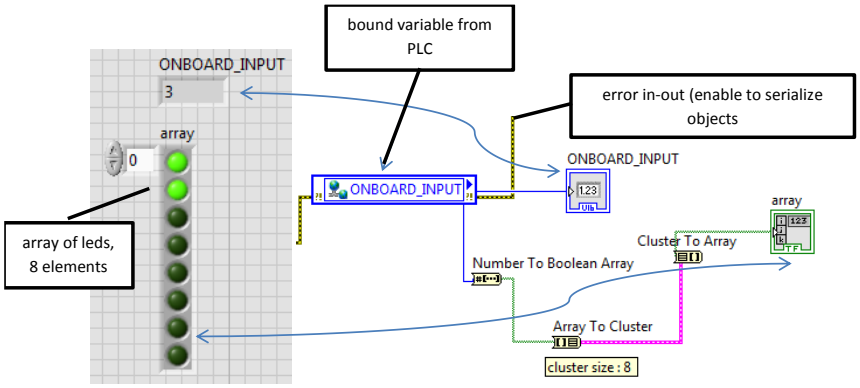
Using OPC Tag Data in LabVIEW

- A VI is used to create a user interface and executable graphical code.
- By default, you see the Front Panel, which is the user interface of the VI. LabVIEW has many built-in UI components, such as graphs, charts, dials, and so on, that you can use to build a powerful, intuitive UI.
- In the VI, select **Window»Show Block Diagram** . The Block Diagram is where you build the behavior of your application.
- Drag and drop the shared variable from the project explorer to the Block Diagram of the VI.
- The shared variable acts as a source of data to other terminals on the Block Diagram.

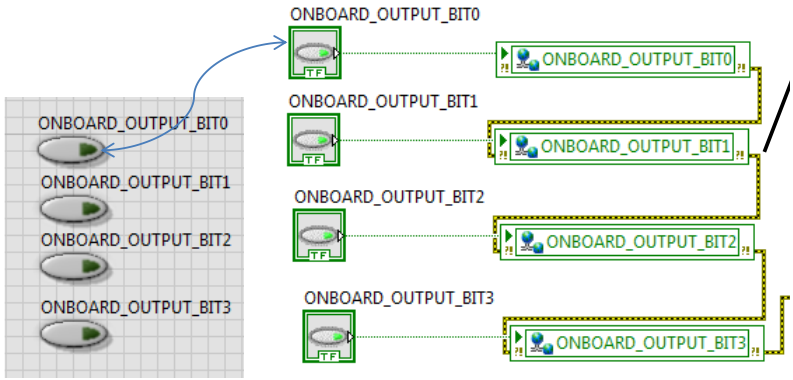
Wiring terminals together on the block diagram

- Select **View»Tools Palette** to show the Tools palette, which contains various tools for building the Block Diagram.
- By default you use the **Automatic Tool Selection** tool, which selects the appropriate tool based on the location of the cursor.
- Select the **Connect Wire** tool. This tool is used to wire terminals together on the Block Diagram.

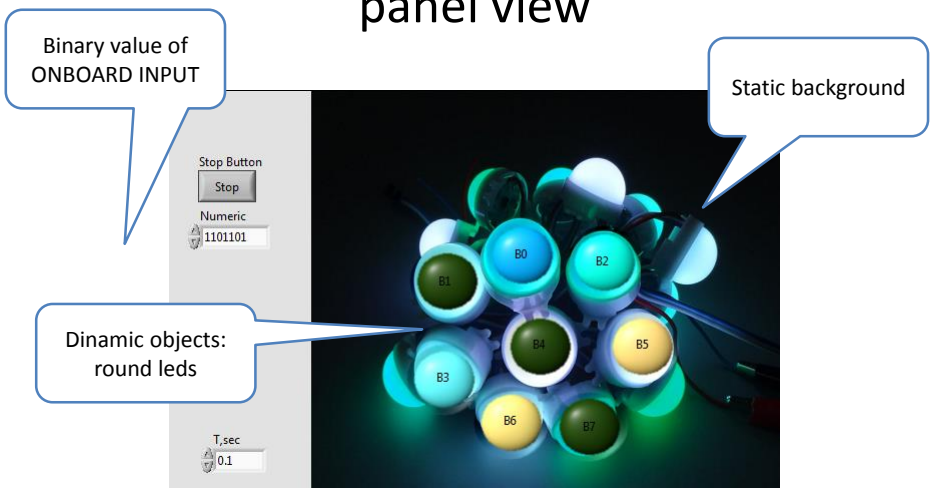
Example: conversion U16 into array of 8 bool elements



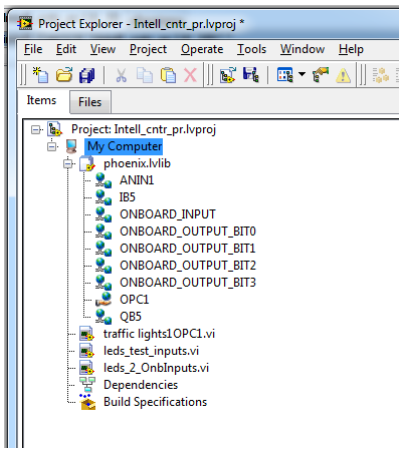
Example: Direct swicth of PLC outputs via bound variables



Leds show states of ONBOARD INPUT in PLC : front panel view

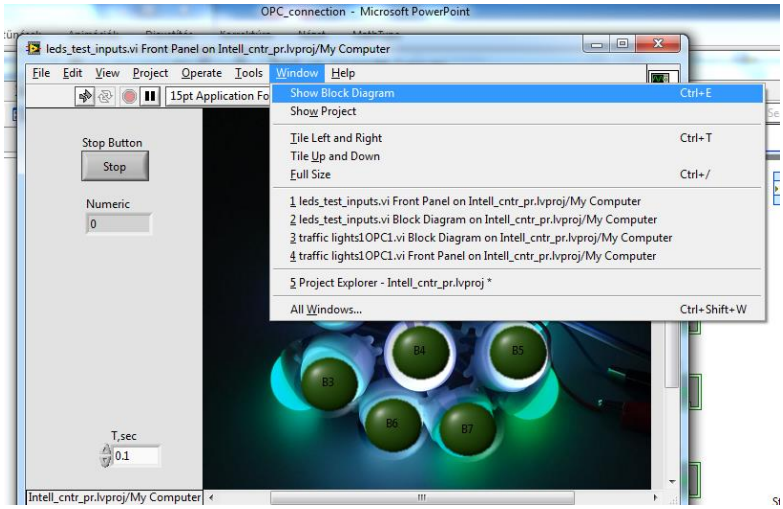


Testing the bounded variables

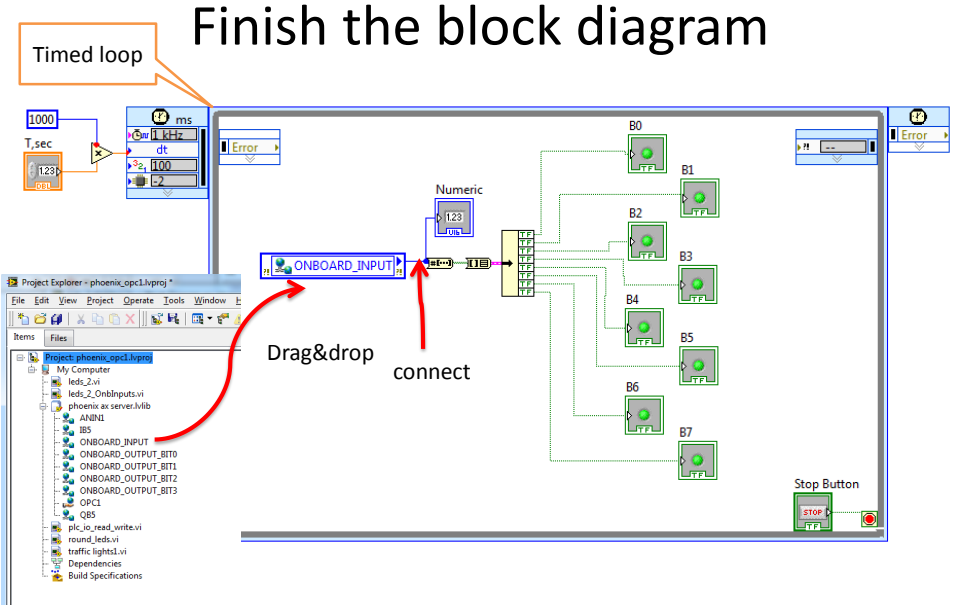


- Open: leds_test_inputs.vi

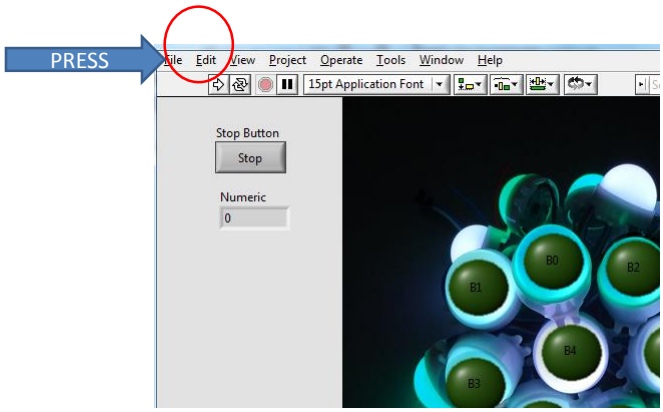
Open the block diagram



Finish the block diagram



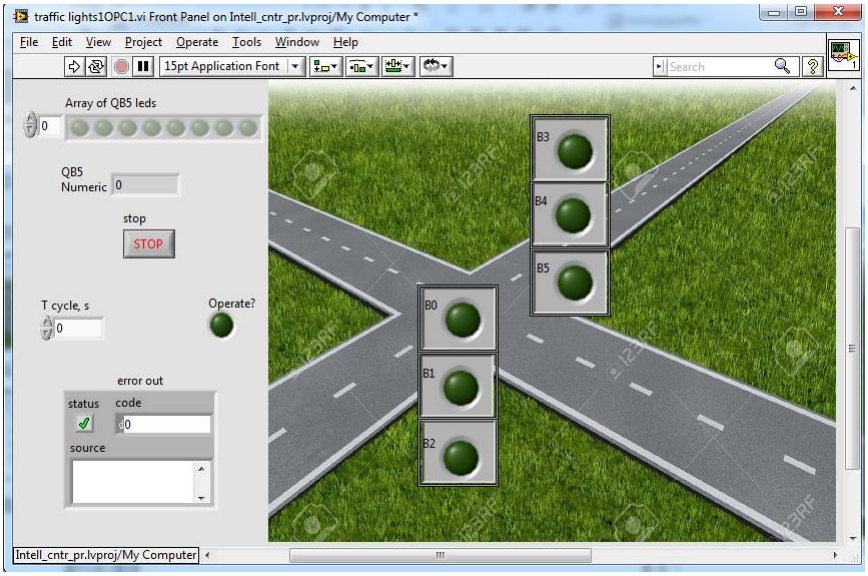
Run the vi,
switch on/off the PLC inputs



Test your traffic light program

- The traffic light program must be run on PLC
- Close the „Project Control Dialog” window
- Open in LabVIEW project the „traffic lights1OPC1.vi”
- In the block diagram drag&drop and wire the bound variables, see next figures

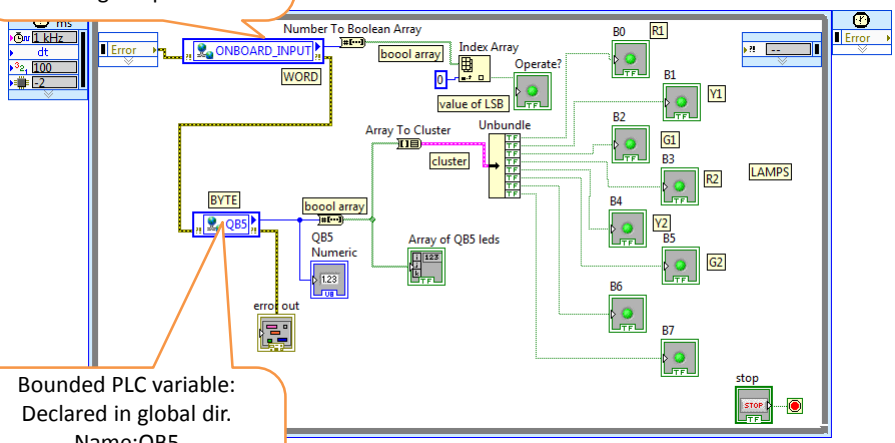
Traffic light control



Block diagram

Bounded PLC variable:
Declared in global dir.
Name:ONBOARD INPUT
Type: word
Bits of Digit. inputs of PLC

ONBOARD INPUT BIT0 switch on/off the traffic control



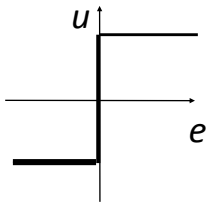
Bounded PLC variable:
Declared in global dir.
Name:QB5
Type: Byte
Address: %QB5
Bits of Digit. output modul

CLASSIC CONTROL: ON-OFF CONTROL

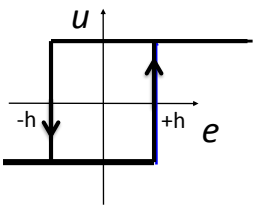
On-Off control

- On-Off control or two-position control system is the simplest form of feedback control.
- The actuating element has only two fixed positions, which are, in many cases, simply on and off.
- The on–off control is relatively simple and inexpensive therefore is very widely used in both industrial and domestic control systems.

Static characteristics of the on–off controllers



Ideal on-off control



modifications for hysteresis

On off control algorithms

- No hysteresis :
- $u(t = nT) = \begin{cases} u_{min}, if\ e < 0 \\ u_{max}, if\ e \geq 0 \end{cases}$
- With hysteresis:
- $u(t = nT) = \begin{cases} u_{min}, if\ e < -h \\ u_{max}, if\ e \geq h \\ no\ change\colon\ u(t = (n-1)T) \end{cases}$

On/Off Control Has Limited Use

- On/off with dead band is useful for home appliances such as furnaces, air conditioners, ovens & refrigerators
- For most industrial applications, on/off is too limiting

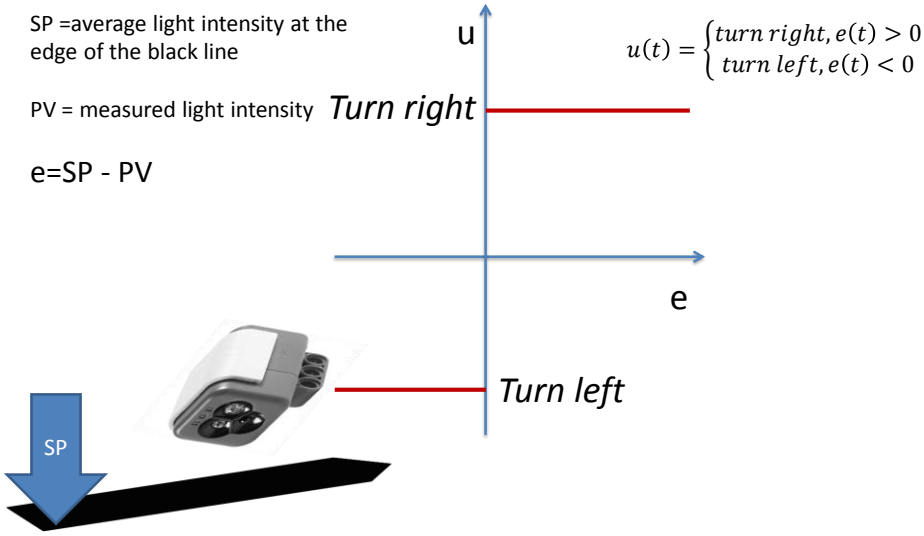
Think about riding in a car that has on/off cruise control

On-off controller for a line following robot

SP =average light intensity at the edge of the black line

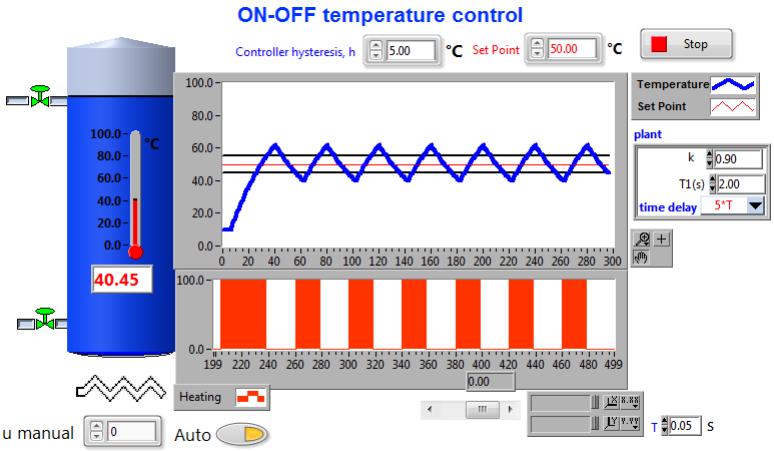
PV = measured light intensity

e=SP - PV

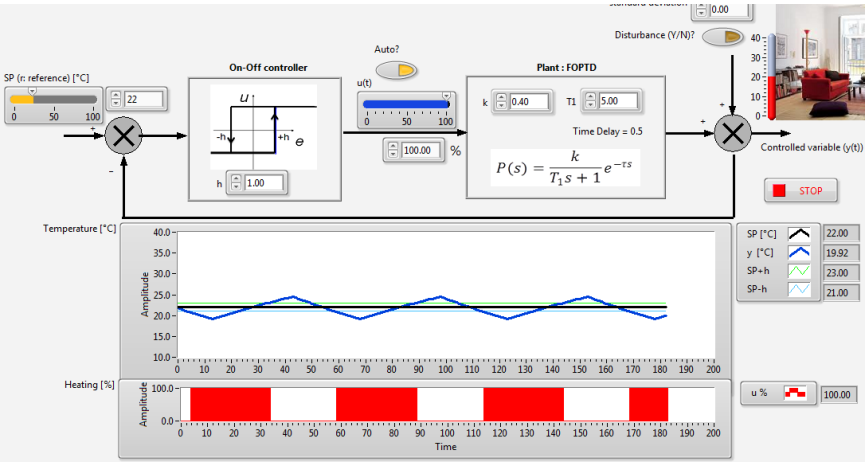


Line Follower on off control

On-off control of a first-order plus time delay plant

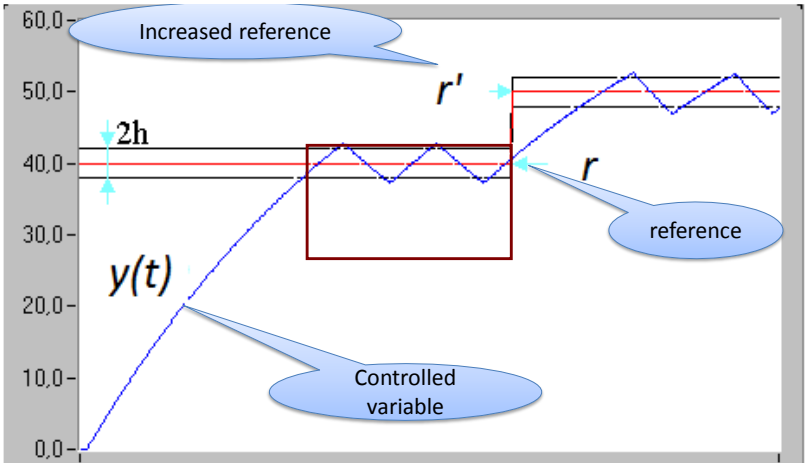


Example on-off control of room temperature



Quasi-stationary state

- In a steady state, the characteristic of the controlled variable performs a constant amplitude oscillation around a mean value
- For nonlinear systems, the quasi-stationary state is a stable state.
- Because of the controller characteristic, the on-off control loop is a nonlinear system

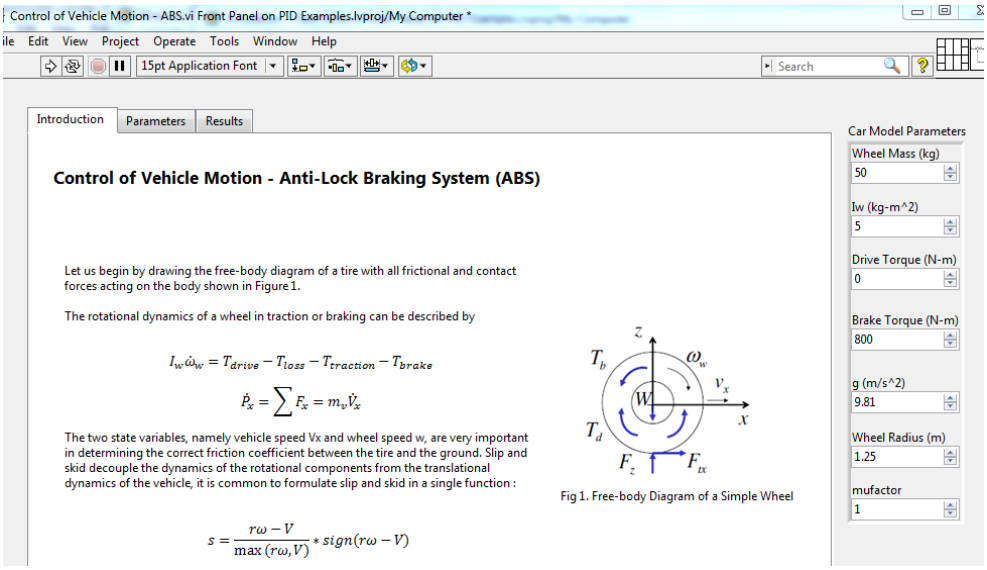


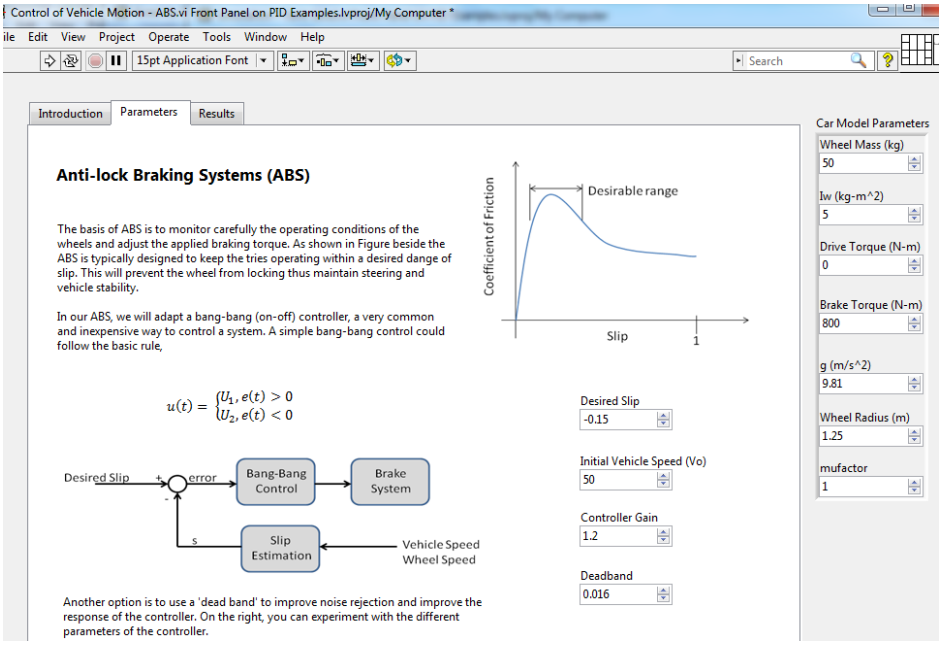
Increasing reference: r' > r

$$\Delta y^*_{Off} < \Delta y_{Off} \qquad \Delta y^*_{On} > \Delta y_{On}$$

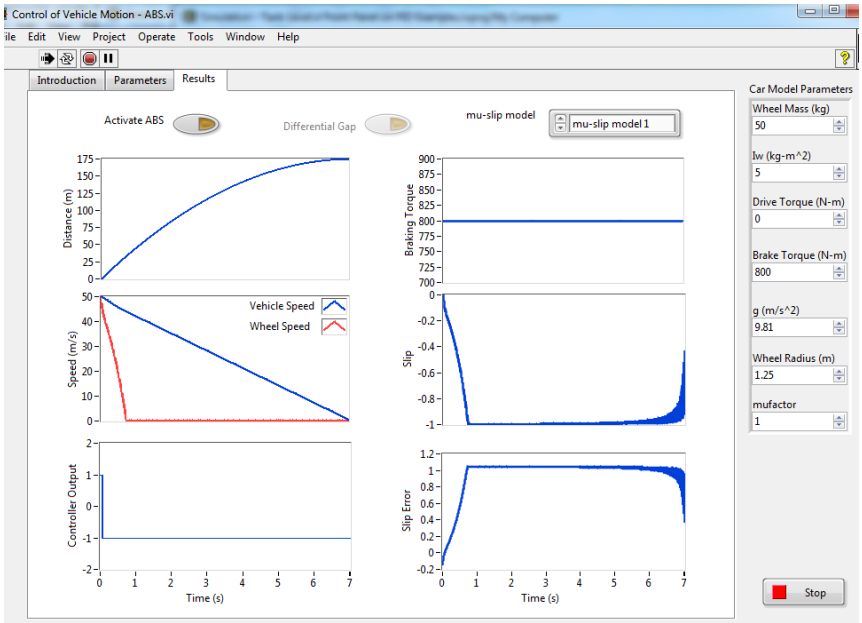
$$\frac{T'_b}{T'_k} > \frac{T_b}{T_k}$$

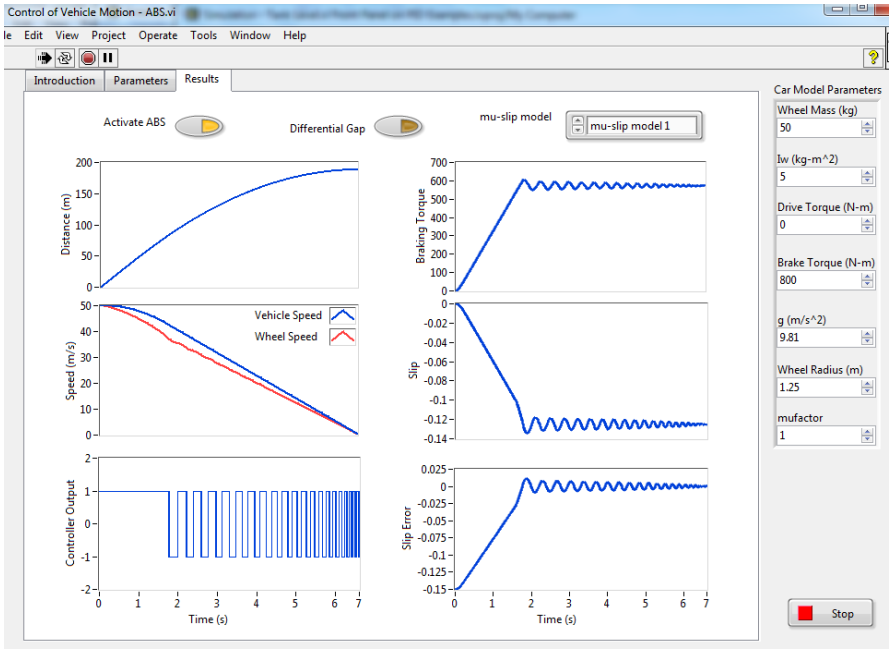
ABS



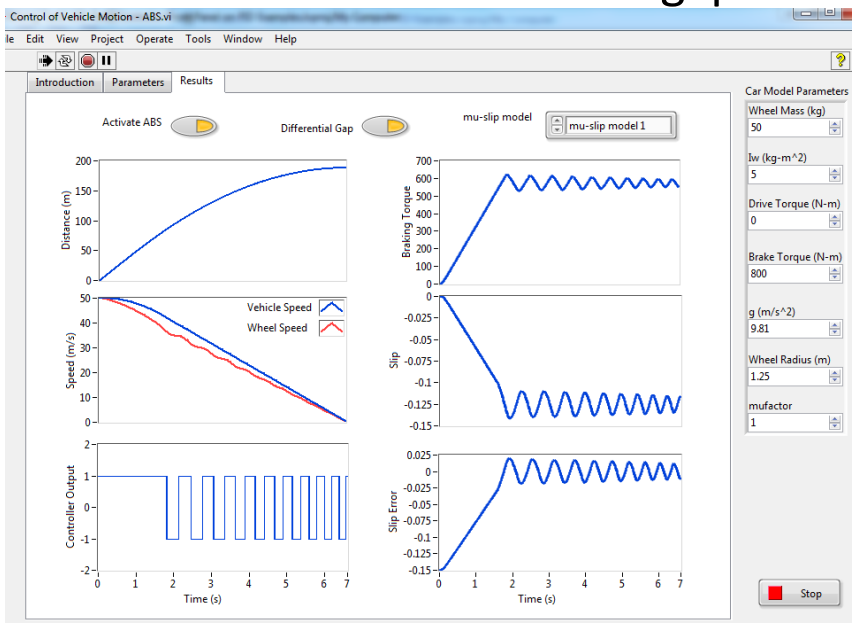


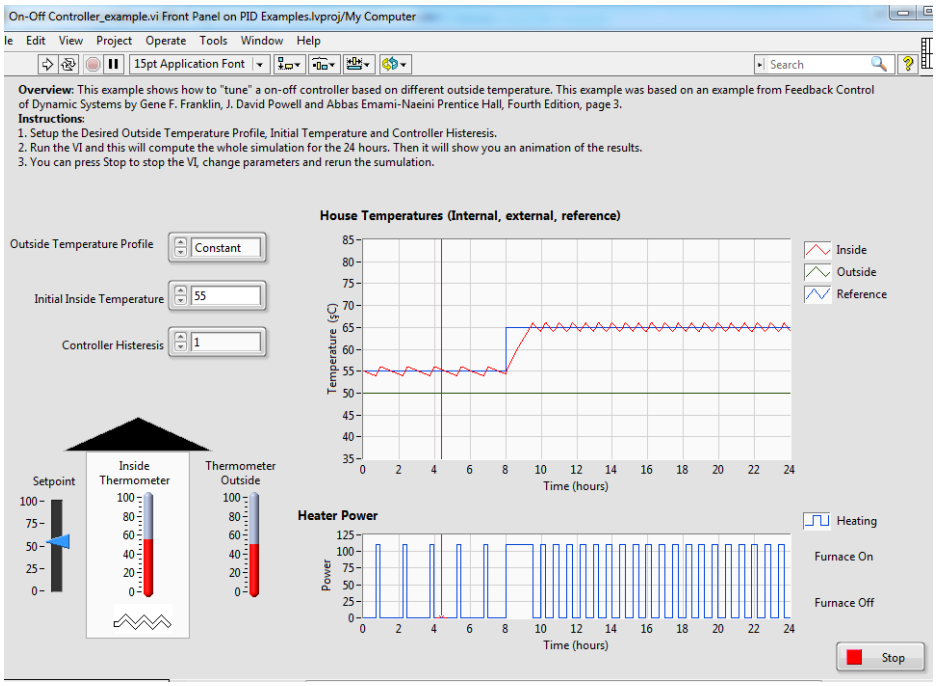
Without ABS control

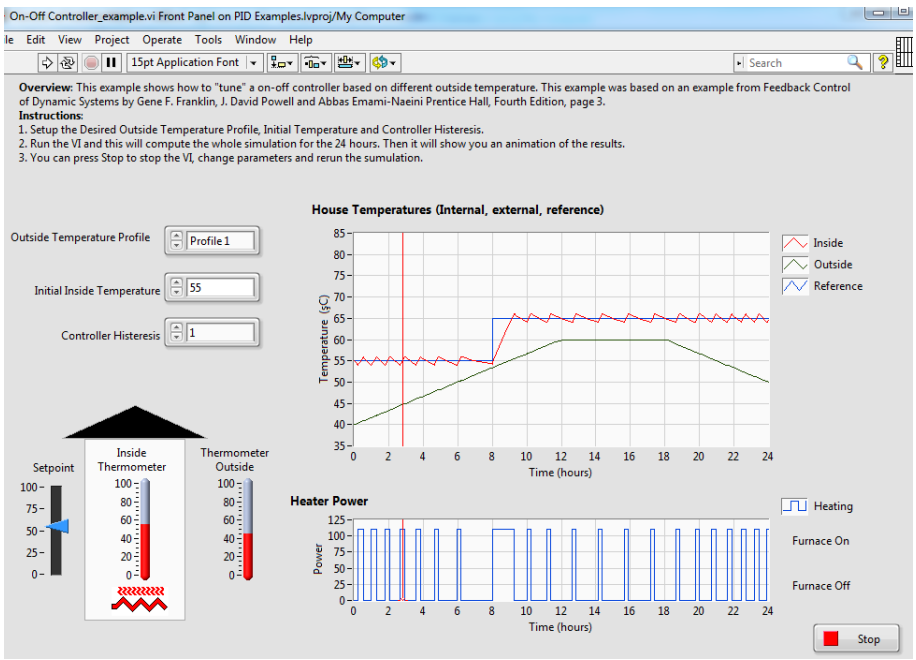


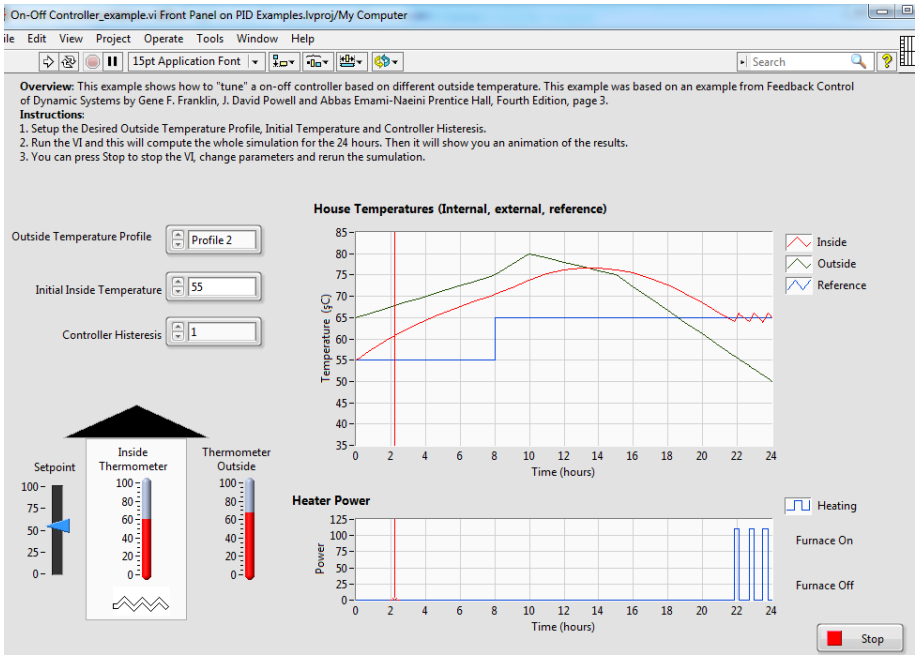


With ABS & differential gap




On-Off Controller_example.vi Front Panel on PID Examples.lvproj/My Computer
Edit View Project Operate Tools Window Help
15pt Application Font
Search
Overview: This example shows how to "tune" a on-off controller based on different outside temperature. This example was based on an example from Feedback Control of Dynamic Systems by Gene F. Franklin, J. David Powell and Abbas Emami-Naeini Prentice Hall, Fourth Edition, page 3.
Instructions:
1. Setup the Desired Outside Temperature Profile, Initial Temperature and Controller Histeresis.
2. Run the VI and this will compute the whole simulation for the 24 hours. Then it will show you an animation of the results.
3. You can press Stop to stop the VI, change parameters and rerun the sumulation.
Outside Temperature Profile
Constant
Initial Inside Temperature
55
Controller Histeresis
1
House Temperatures (Internal, external, reference)
Temperature (ºC)
Time (hours)
Inside
Outside
Reference
Setpoint
Inside Thermometer
Thermometer Outside
Heater Power
Power
Heating
Furnace On
Furnace Off
Stop

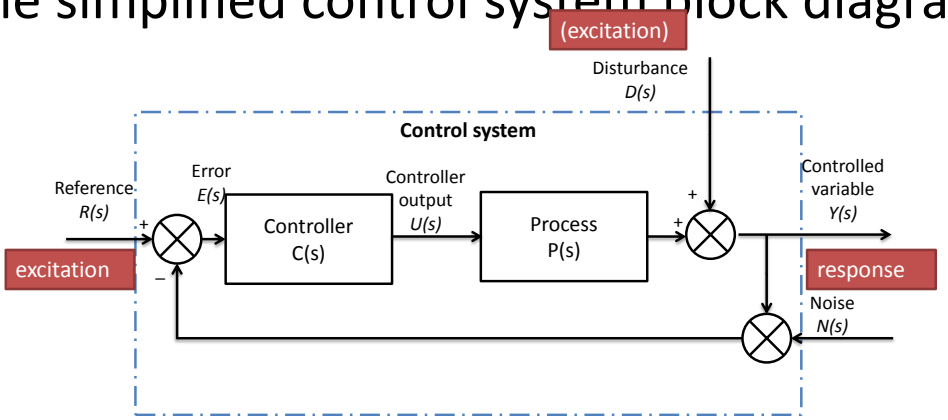

On-Off Controller_example.vi Front Panel on PID Examples.lvproj/My Computer
Edit View Project Operate Tools Window Help
15pt Application Font
Search
Overview: This example shows how to "tune" a on-off controller based on different outside temperature. This example was based on an example from Feedback Control of Dynamic Systems by Gene F. Franklin, J. David Powell and Abbas Emami-Naeini Prentice Hall, Fourth Edition, page 3.
Instructions:
1. Setup the Desired Outside Temperature Profile, Initial Temperature and Controller Histeresis.
2. Run the VI and this will compute the whole simulation for the 24 hours. Then it will show you an animation of the results.
3. You can press Stop to stop the VI, change parameters and rerun the sumulation.
Outside Temperature Profile
Profile 1
Initial Inside Temperature
55
Controller Histeresis
1
House Temperatures (Internal, external, reference)
Temperature (ºC)
Time (hours)
Inside
Outside
Reference
Setpoint
Inside Thermometer
Thermometer Outside
Heater Power
Power
Heating
Furnace On
Furnace Off
Stop



Intelligent control systems

CLASSIC CONTROL: PID CONTROL

The simplified control system block diagram



Intelligent control systems

CONTROL SYSTEM DESIGN REQUIREMENTS AND SPECIFICATION

Control system design requirements and specification

The requirement can be arranged in the next four main groups:

Stability.

- The negative feedback has the potential risks of instability. Delays and high-gain factors cause instability in a closed loop.
- The stability of the control loop to be asymptotic stability:

  $\lim_{t\to\infty} y_0(t) = 0$, where $y_0(t)$ is the zero-input response.

- Suitable static accuracy of disturbance suppression and/or setpoint tracking.
- Prescribed dynamic behavior.
- Insensitivity for process parameter changes. (Robustness.)

Transient response and steady-state response

The time response of a control system consists of two parts: the transient response and the steady-state response.

The transient response is defined as the part of the time response which goes from the initial state to the final state and reduces to zero as time becomes very large.

The steady-state response is defined as the behavior of the system as t approaches infinity, $t \to \infty$, after the transients have died out.

asymptotic stability: $\lim_{t\to\infty} y_0(t) = 0$, where $y_0(t)$ is the zero-input response.

Asymptotic stability of the feedback system with the response function :

$\lim_{t\to\infty} y_{tr}(t) = 0.$

Therefore the system response depends only on the steady-state component if the elapsed time is large enough:

$\lim_{t\to\infty} y(t) = y_{ss}(t).$

$$y(t) = y_{tr}(t) + y_{ss}(t)$$

Static accuracy of disturbance suppression and/or set-point tracking

It is desired that the controlled variable follows the reference change asymptotically:

$$\lim_{t\to\infty} e(t) = \lim_{t\to\infty} \big(r(t) - y(t)\big) = r(\infty) - y_{ss} = 0$$

the steady-state approximates the reference:

$$y_{ss} \cong r(\infty)$$

The stationary behavior of a control loop is provided: be the static component equal to the reference!

(If not, remaining error: the steady-state error !!)

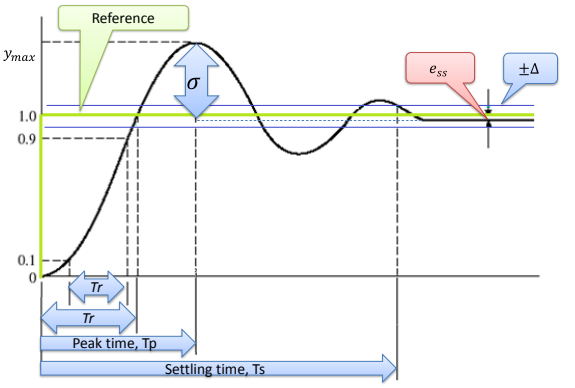
It should be specified that for what type of input signals should be required to produce the reference signal the steady-state behavior.

Usually the steady-state criteria are prescribed for step like input changes:

$$e(\infty) = e_{ss} = r - y_{ss} = 0$$

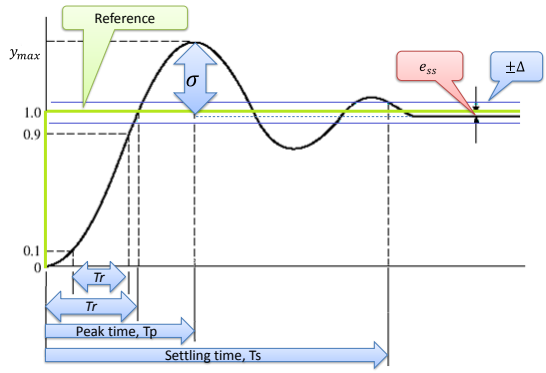
Prescribed dynamic behavior: Performance meters for unit step reference response

- The transient response of a system may be described using two factors, the swiftness and the closeness of the response to the desired response.
- The swiftness of the response is measured by the rise time (T_r) and the peak time (T_p). The closeness is measured by the overshoot (σ) and settling time (T_s).



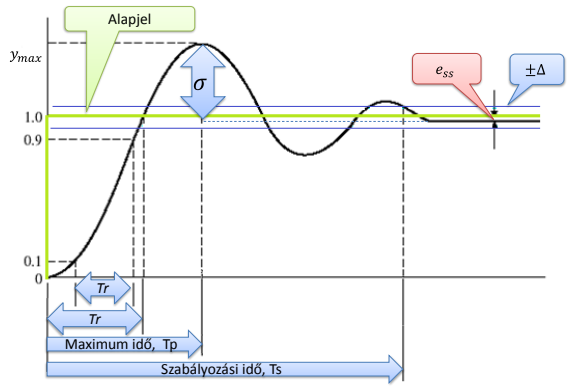
Performance meters for unit step reference response

- *Rise Time*, T_r: The time required for the waveform to go from 0/0.1 of the final value to 1/0.9 of the final value. (At underdamped systems: 0-100% rise time is used; at overdamped systems: 10-90% rise time is used.)



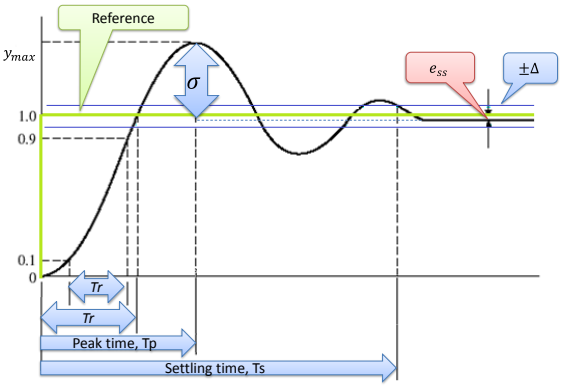
Performance meters for unit step reference response

- *Peak Time*, T_p. The time required to reach the first, or maximum, peak.
- *Percentage Overshoot*: $S\% = \frac{y(T_p)-y(\infty)}{y(\infty)}100\% = \frac{\sigma}{y(\infty)}100\%$. $y(0) = 0$. The amount that the waveform overshoots the steady-state, or final value at the peak time, expressed as a percentage of the steady-state value.



Performance meters for unit step reference response

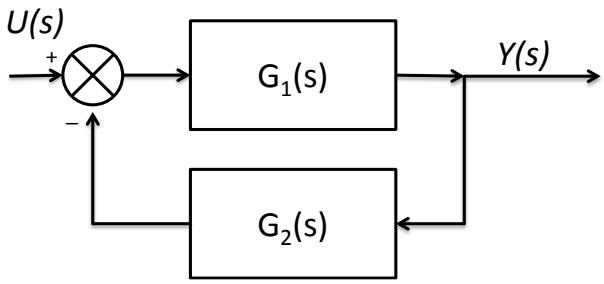
- *Settling Time*, T_s. The time required for the transient's damped oscillations to reach and stay within $\pm\Delta\%$ (±5% or ±2%) of the new steady-state value.
- *Error band or tracking band*, Δ. Required to determining the settling time: $y(\infty)(1-\Delta) < y(t \geq T_s) < y(\infty)(1+\Delta)$.



Intelligent control systems

CONTROL LOOP ANALYSIS IN THE FREQUENCY DOMAIN

Negative feedback



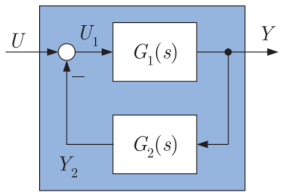
Closed loot transfer functions:

$$G(s) = \frac{G_1(s)}{1 + G_1(s)G_2(s)}$$

The input – output Laplace-transform relation:

$$Y(s) = \frac{G_1(s)}{1 + G_1(s)G_2(s)} U(s)$$

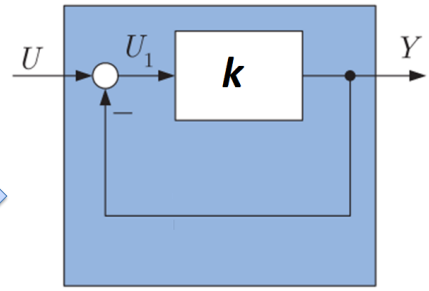
The effect of the negative feedback on the system gain



$$G_v(s) = \frac{G_1(s)}{1 + G_1(s)G_2(s)}$$

simplify the system

$$G_1(s) = k$$
$$G_2(s) = 1$$

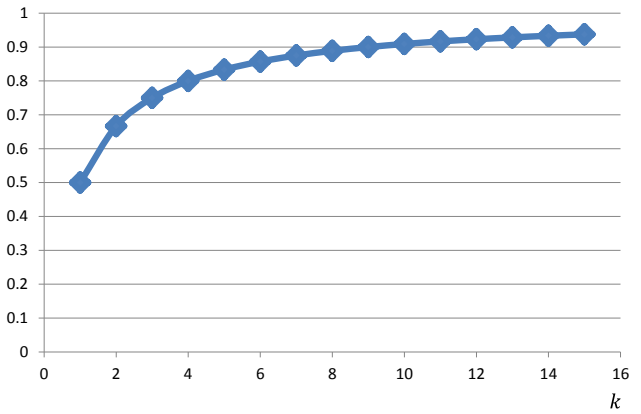


The system transfer function

$$G(s) = \frac{G_1(s)}{1 + G_1(s)} = \frac{k}{1 + k} = k^*$$

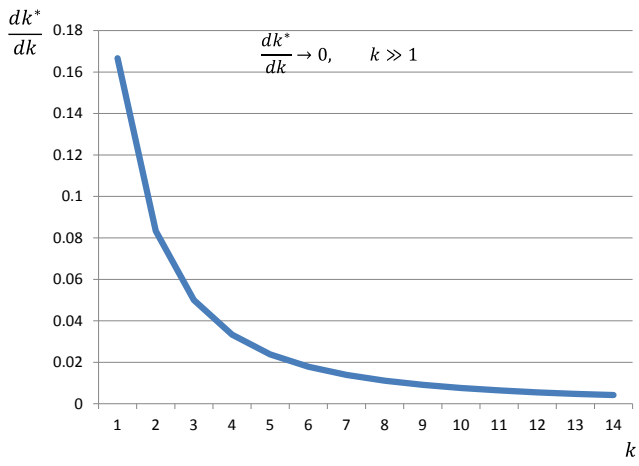
The closed-loop gain versus the system gain

$$k^* = \frac{k}{1+k}$$

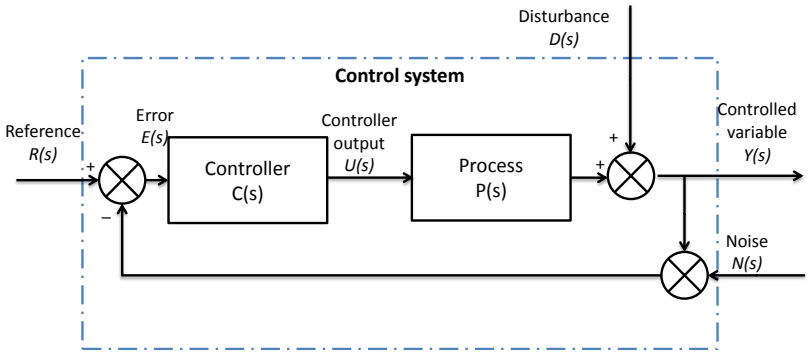


With increasing value of *k*, the sensitivity of the closed-loop gain to the system gain is decreases, even at not too high *k* approaches to zero.

Sensitivity of a closed-loop gain

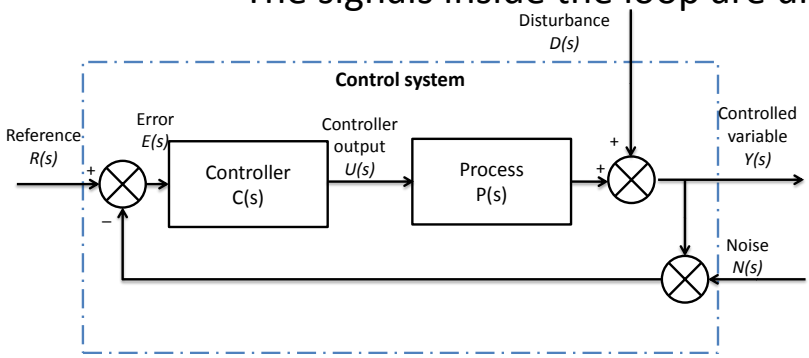


Transfer characteristic of the closed-loop control



The transition behavior of this control loop is specified according to the tree inputs (command, disturbance and noise): reference tracking or command behavior; disturbance behavior and noise behavior.

- Suppose : C(s) and P(s) are known.
- Independent variables: inputs, dependent variable: contolled variable.
- The signals inside the loop are unknown.



- Laplace-transform of the controlled variable:

$$Y(s) = P(s)U(s) + D(s)$$

U(s) unknown, must be rejected:

$$U(s) = C(s)\Big(R(s) - \big(Y(s) + N(s)\big)\Big)$$

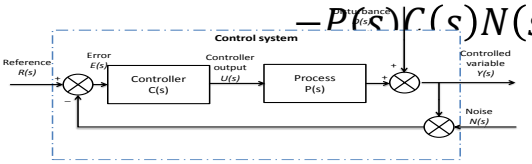
substituting:

$$Y(s) = P(s)C(s)\Big(R(s) - \big(Y(s) + N(s)\big)\Big) + D(s)$$

rearranging:

$$Y(s) = P(s)C(s)R(s) - P(s)C(s)Y(s) - P(s)C(s)N(s) + D(s)$$

$$Y(s)\big(1 + P(s)C(s)\big) = P(s)C(s)R(s) - P(s)C(s)N(s) + D(s)$$



The resulting relationship between the output signal and the Laplace-transform of the input signals

$$Y(s) = \frac{C(s)P(s)}{1 + C(s)P(s)}R(s) - \frac{C(s)P(s)}{1 + C(s)P(s)}N(s) + \frac{1}{1 + C(s)P(s)}D(s)$$

- for $N(s) = 0$ and $D(s) = 0$, the ***reference tracking transfer function*** can be obtained (the transfer function from $R(s)$ to $Y(s)$):

$$G_R(s) = \frac{Y(s)}{R(s)} = \frac{C(s)P(s)}{1 + C(s)P(s)} = \frac{G_0(s)}{1 + G_0(s)}$$

$$Y(s) = \frac{C(s)P(s)}{1 + C(s)P(s)} R(s) - \frac{C(s)P(s)}{1 + C(s)P(s)} N(s) + \frac{1}{1 + C(s)P(s)} D(s)$$

- For $R(s) = 0$ és $N(s) = 0$, the ***disturbance transfer function*** is obtained (the transfer function from $D(s)$ to $Y(s)$):

$$G_D(s) = \frac{Y(s)}{D(s)} = \frac{1}{1 + C(s)P(s)} = \frac{1}{1 + G_0(s)}$$

$$Y(s) = \frac{C(s)P(s)}{1 + C(s)P(s)} R(s) - \frac{C(s)P(s)}{1 + C(s)P(s)} N(s) + \frac{1}{1 + C(s)P(s)} D(s)$$

- for $R(s) = 0$ and $D(s) = 0$, the ***noise transfer function*** is obtained (the transfer function from $N(s)$ to $Y(s)$):

- $G_N(s) = \frac{Y(s)}{N(s)} = -\frac{C(s)P(s)}{1+C(s)P(s)} = -\frac{G_0(s)}{1+G_0(s)}$

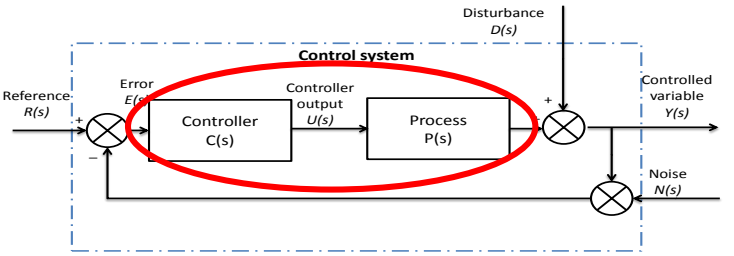
Relationship between the output signal and the Laplace-transform of the input signals

The (open) loop transfer function

$$G_0(s) = C(s)P(s).$$

The open-loop gain (derived from the Laplace-transform initial value theorem):

$$G_0(0) = C(0)P(0) = k_0$$



The closed-loop transfer functions

The closed−loop input – output relation in frequency− or Laplace−transform domain is

$$Y(s) = G_R R(s) + G_N N(s) + G_D D(s)$$

$$Y(s) = \frac{C(s)P(s)}{1 + C(s)P(s)} R(s) - \frac{C(s)P(s)}{1 + C(s)P(s)} N(s) + \frac{1}{1 + C(s)P(s)} D(s)$$

For the system to be stable all transfer functions must be stable!

Stability
of the feedback control system

a necessary and sufficient condition for a feedback system to be stable is that all the poles of the system transfer function have negative real parts

$$Y(s) = \frac{C(s)P(s)}{1 + C(s)P(s)} R(s) - \frac{C(s)P(s)}{1 + C(s)P(s)} N(s) + \frac{1}{1 + C(s)P(s)} D(s)$$

Notice that the denominators of the transfer functions are same.

Since the stability depends on the poles of the transfer function, it is enough to analyze stability for one of the inputs, for example for the reference tracking, if it proves to be stable, the control loop is stable for all remaining inputs as well.

Stability
of the feedback control system

The denominator of the closed-loop transfer functions is:

$1 + C(s)P(s)$ or $1 + C(j\omega)P(j\omega)$.

A system is stable if all the poles of the system transfer function have negative real parts. In other words: *there is no positive frequency*, for which

$1 + C(j\omega)P(j\omega) = 0$ or $C(j\omega)P(j\omega) = -1$.

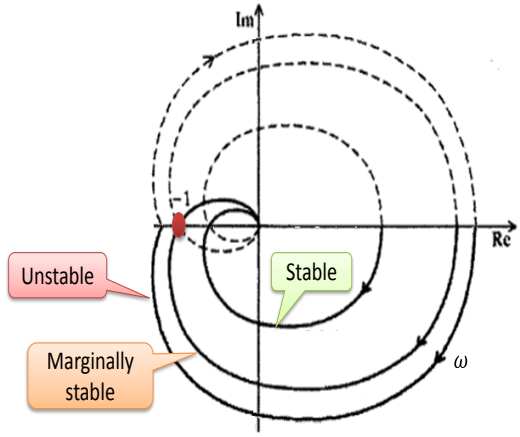
This means that instead of examining the characteristic equation (the denominator) poles, enough to analyze the loop transfer function equality with -1.

The system is stable if the loop transfer function, $G_0(j\omega)$ is *not equal* -1 at any positive frequency: $G_0(j\omega) = C(j\omega)P(j\omega) \neq -1$.

The stability of the closed loop can be deduced from the transfer function of the open loop!

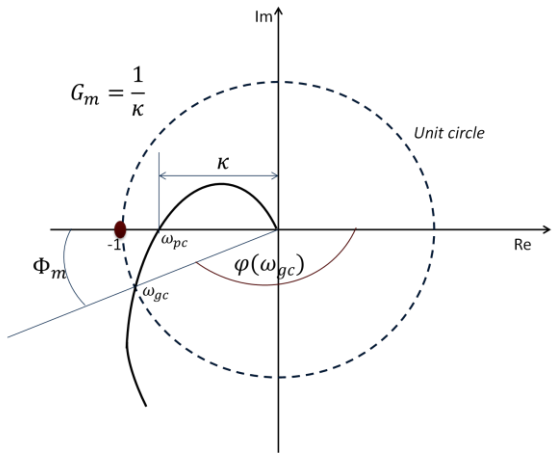
The Nyquist stability criterion

- Is a graphical technique for determining the stability of a dynamical system. Because it only looks at the Nyquist plot of the *open-loop* systems, it can be applied without explicitly computing the poles and zeros of either the closed-loop or open-loop system
- it is restricted to linear, time-invariant (LTI) systems.
- the *closed-loop* is asymptotically stable, if the frequency response locus of the *open-loop* does neither revolve around or pass through the critical point (-1,j0).



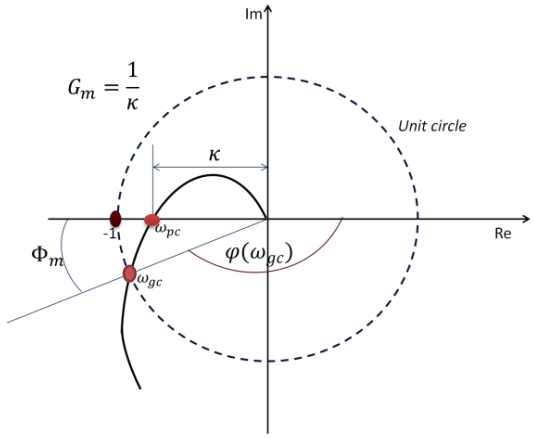
Phase and Gain stability margins

- Using the Nyquist diagram, we can define two quantitative measures of how stable a system is.
- These quantities are called gain margin and phase margin.
- They give the degree of relative stability; in other words, they tell how far the given system is from the instability region.
- Systems with greater gain and phase margins can withstand greater changes in system parameters before becoming unstable.



Crossover frequencies

- ω_{pc} : *phase crossover frequency*: the frequency when the phase $\angle G_0(j\omega_{gc}) = -180°$.
- ω_{gc} : *gain crossover frequency*: the frequency when the gain $|G_0(j\omega_{gc})| = 1$.



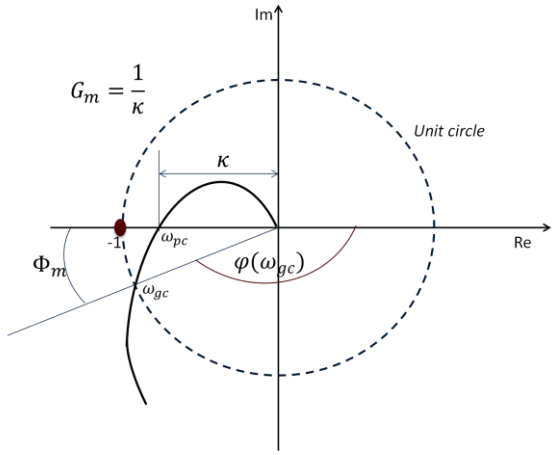
The gain margin, G_m

- The magnitude response of stable systems at the phase crossover frequency:

$|G_0(j\omega_{pc})| = \kappa < 1$.

- The gain margin is the change in open-loop gain, required at 180° of phase shift to make the closed-loop system unstable.
- In the cases of marginally stable systems: $\kappa = 1$.

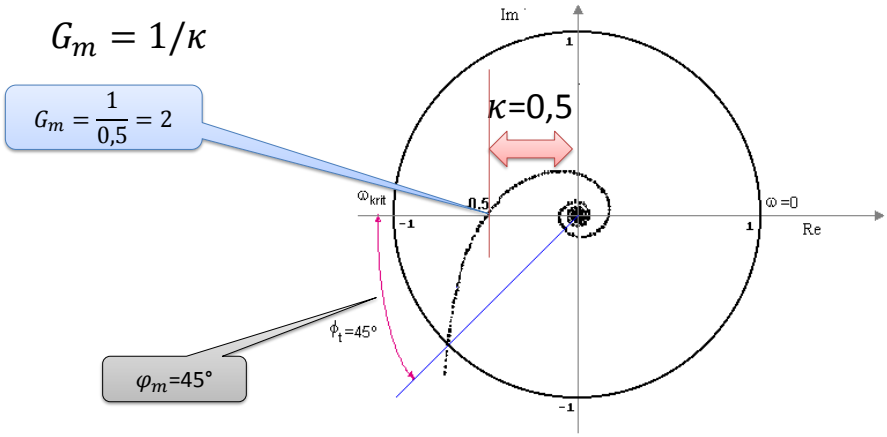
$$G_m = 1/\kappa$$



The gain margin, G_m

- Example:

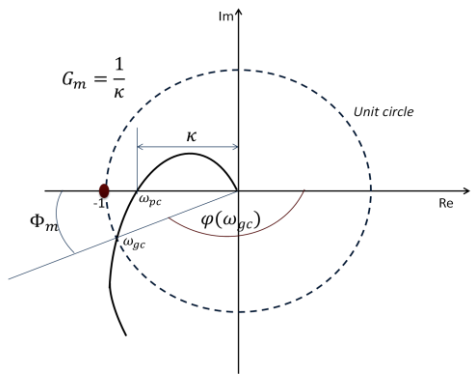
$G_m = 1/\kappa$



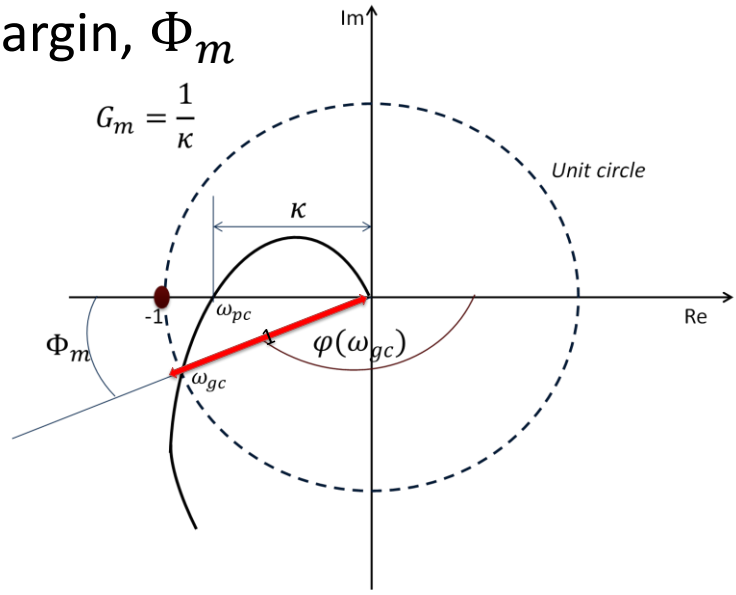
The phase margin, Φ_m

- The phase margin is the change in open-loop phase shift required at unity gain to make the closed-loop system unstable.

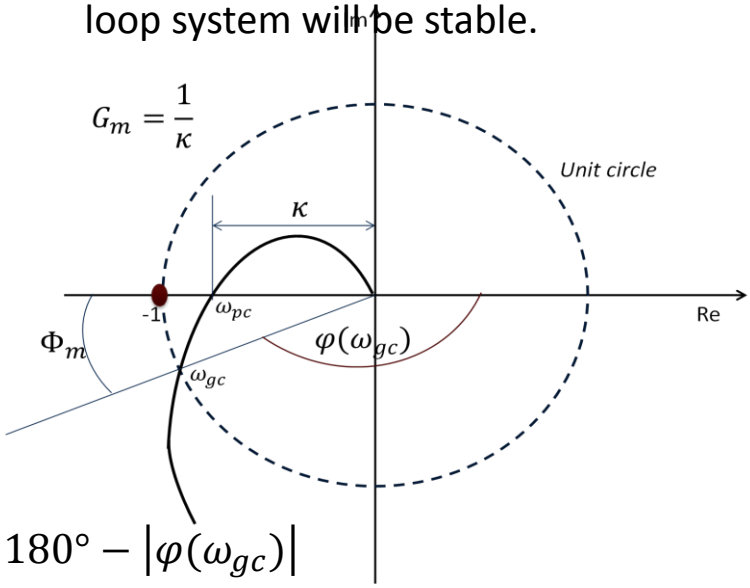
$$\Phi_m = 180° - |\varphi(\omega_{gc})|$$



The phase margin, Φ_m



If the gain margin and phase margin are both positive the closed-loop system will be stable.



$$\Phi_m = 180° - |\varphi(\omega_{gc})|$$

Phase and Gain stability margins in the Bode graphs

- *Bode graphs* are representations of the magnitude and phase of $G_0(j\omega)$ (where the frequency vector ω contains only positive frequencies).
- These consist of two graphs. The first plot is a plot of

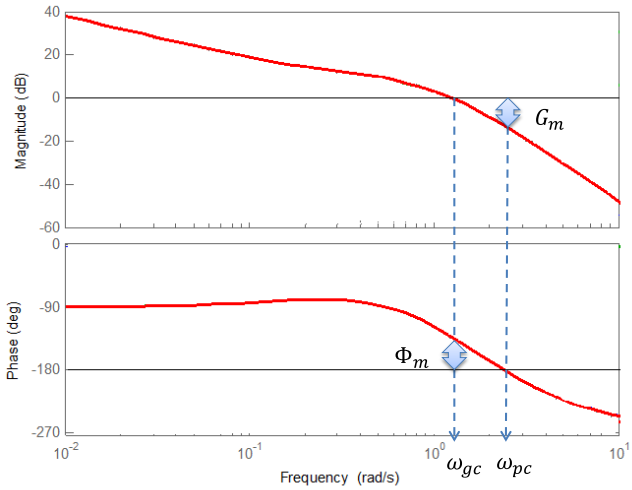
log |G(j ω)| (in decibels, dB) versus frequency; referred as *magnitude frequency response*.

- We define the magnitude frequency response to be the ratio of the output sinusoid's magnitude to the input sinusoid's magnitude. (dB : $20\log_{10}|G(j\,\omega)|$)
- The second plot is the phase shift (in degrees) versus frequency (ω), referred as *phase response. The* phase shift is the difference in phase angle between the output and the input sinusoids.
- Both responses are a function of frequency and apply only to the steady-state sinusoidal response of the system. Both plots usually have the frequency (ω) in logarithmic scale.

Phase and Gain stability margins in the Bode graphs

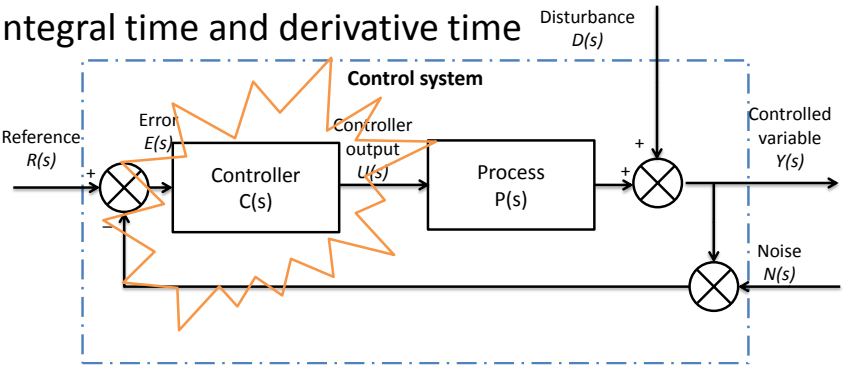
- As 0 dB corresponds to a magnitude of 1, the gain margin is the number of dB that $|G(j\omega_{pc})|$ is *below* 0 dB at the frequency when the phase 180^o (ω_{pc}, phase crossover).
- The phase margin in the number of degrees $\arg(G(i\omega_{gc}))$ is *above* -180^o frequency associated with a gain $|G(j\omega_{gc})|= 1$ (0 dB) (ω_{gc}, gain crossover).
- If the gain margin and phase margin are both positive the system will be stable.

Phase and Gain stability margins in the Bode graphs



Classic controller: PID controller

- The controller structure is given
- The output of the controller depends on the *error*
- Control engineer sets the controller parameters: gain, integral time and derivative time



The I/O modell of the PID-controller :

$$u(t) = K_P e(t) + K_i \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt}$$

Proportional with error: P-effect

The speed of output change is proportional to the error: I-effect

Proportional with the speed of error : D-effect

By aggregating these effects, we get the so-called PID controller.

K_P	Proportional gain of the controller
K_i	Integral gain of the controller
K_D	Derivative gain of the controller

Transfer function of the PID controller

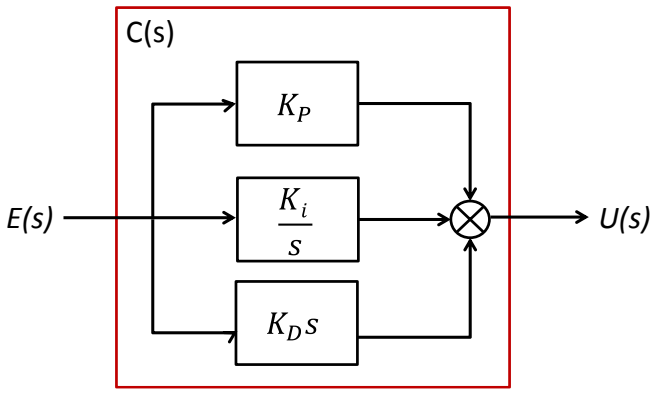
In time domain:

$$u(t) = K_P e(t) + K_i \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt}$$

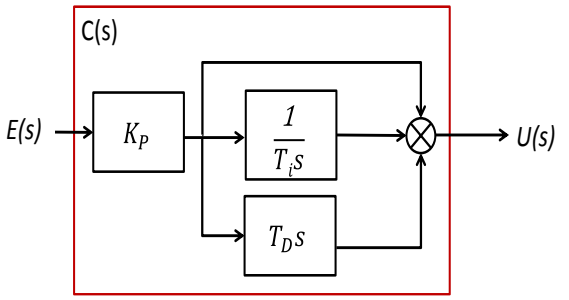
In Laplace domain:

$$C(s) = \frac{U(s)}{E(s)} = K_P + \frac{K_i}{s} + K_D s$$

Structure of parallel PID-controller (gain transfer form)



Time constant representation of the PID-controller



$$C(s) = K_P(1 + \frac{1}{T_i s} + T_D s)$$

K_P	Proportional gain of the controller
T_i	Integral time constant
T_D	Derivative time constant

Time constant representation of the PID-controller (academic form)

I/O model:

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

K_P	Proportional gain of the controller
T_i	Integral time constant
T_D	Derivative time constant

The parallel form and the academic form can be converted to each other

Gain transfer form:

$$u(t) = K_P e(t) + K_i\int_0^t e(\tau)d\tau + K_D\frac{de(t)}{dt}$$

Time constant form:

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

$$u(t) = K_P e(t) + \frac{K_P}{T_i}\int_0^t e(\tau)d\tau + K_P T_D\frac{de(t)}{dt}$$

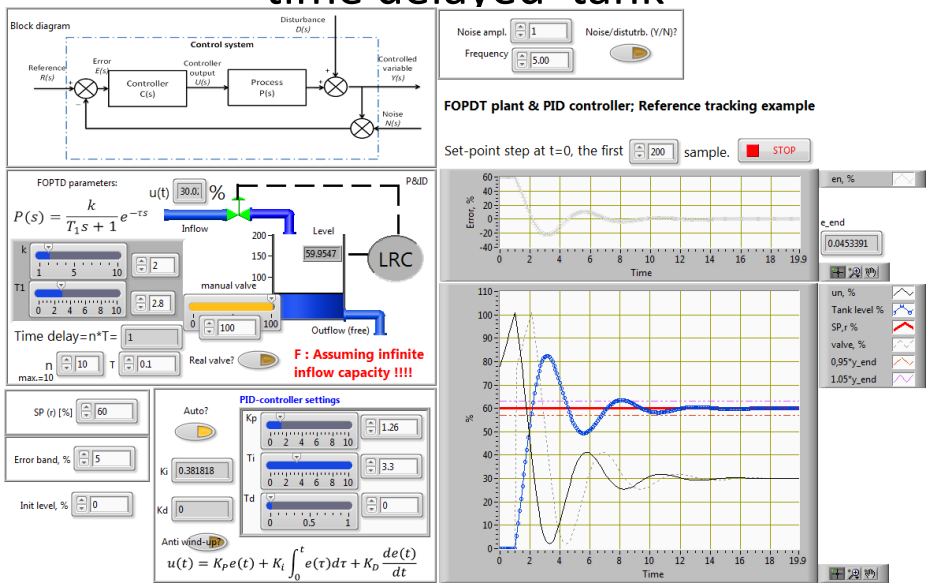
$K_i = \frac{K_P}{T_i}$ *and* K_D=$K_P T_D$

Simpler controller can be derived from PID-controller by appropriate parameterization

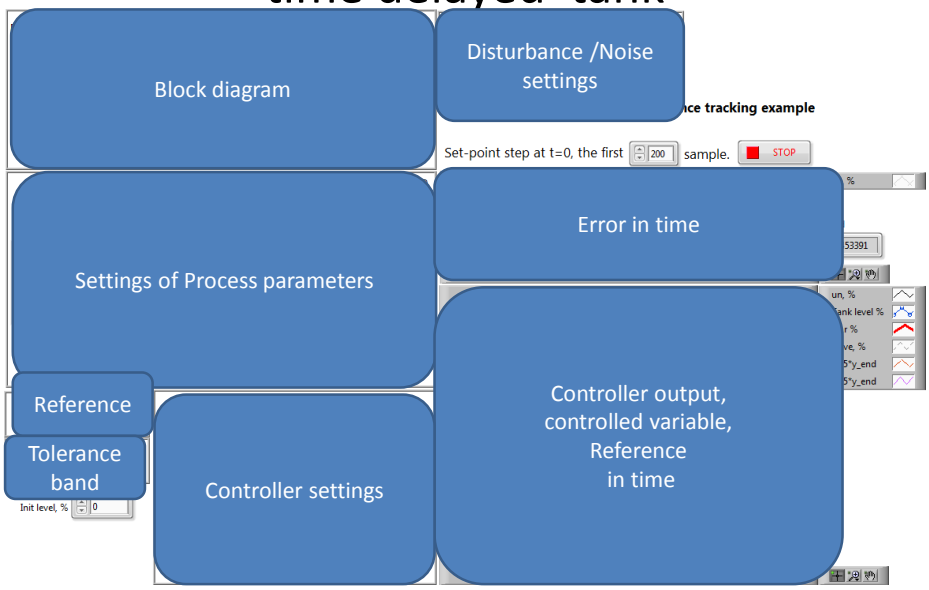
Controller type	Transfer function	PID parameterization
P-	$C(s) = K_P$	$T_i = \infty,\ T_D = 0$
I-	$C(s) = \frac{K_i}{s}$ or $C(s) = \frac{1}{T_i s}$	$K_P = 0,\ K_D = 0$
PI-	$C(s) = K_P\left(1 + \frac{1}{T_i s}\right)$	$T_D = 0$
PD-	$C(s) = K_P(1 + T_D s)$	$T_i = \infty$

$$C(s) = K_P\left(1 + \frac{1}{T_i s} + T_D s\right)$$

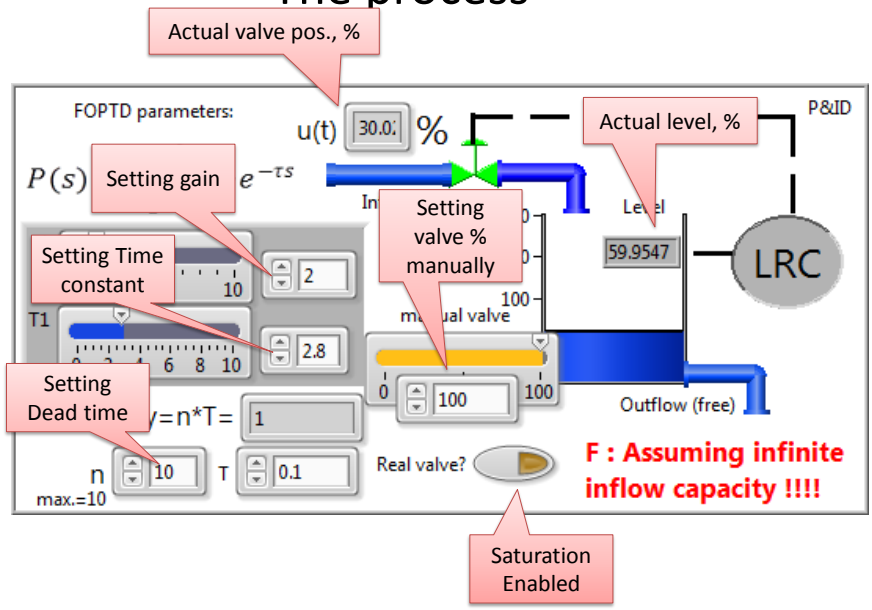
Simulation example: level control of a first-order, time delayed tank



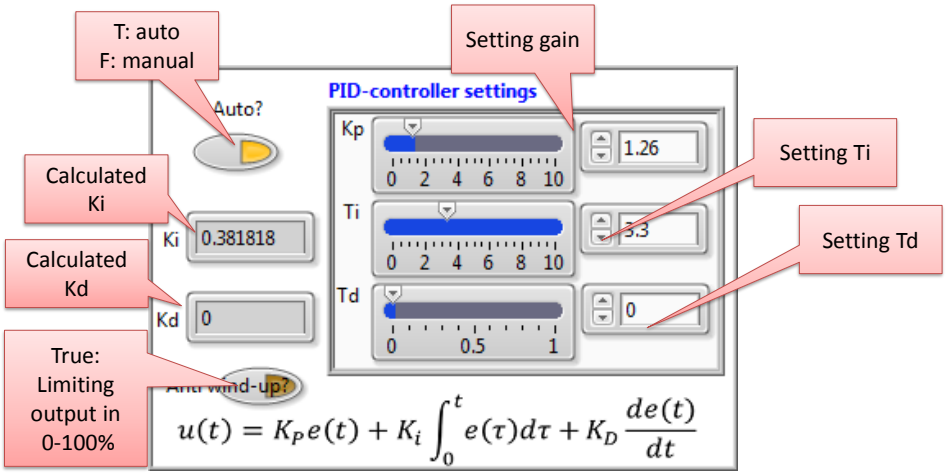
Simulation example: level control of a first-order, time delayed tank



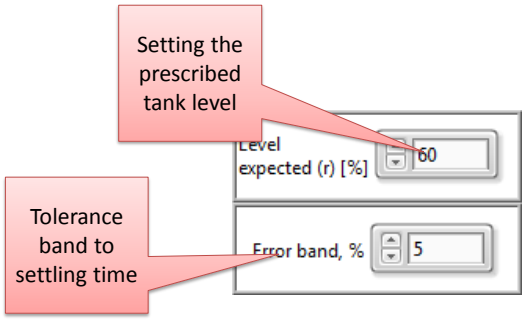
The process



The controller

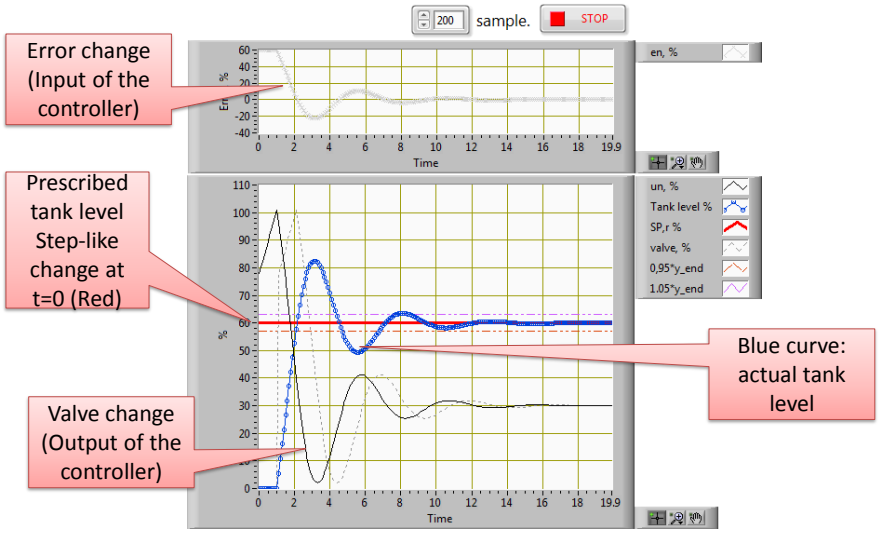


Setpoint (referenece)

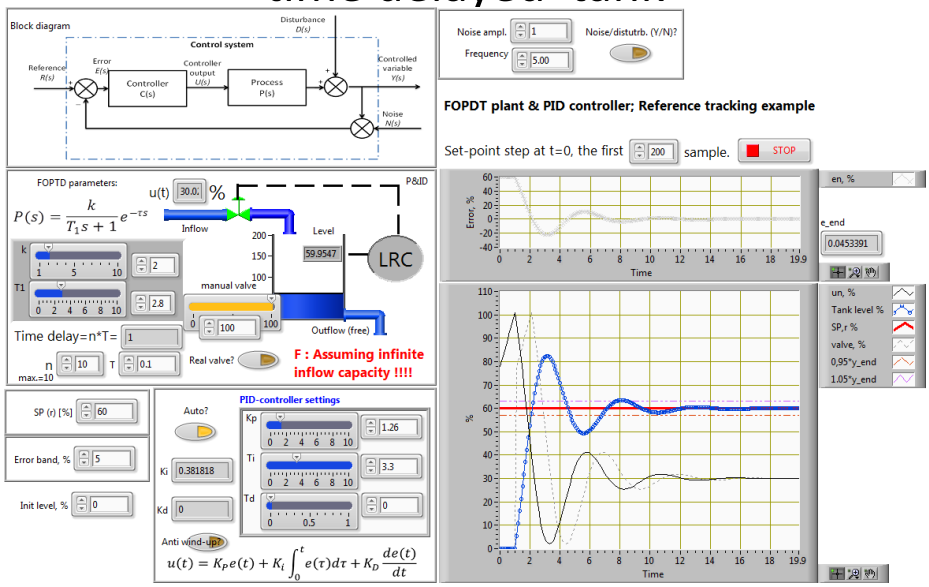


Simulation: At t=0 set point is changing step like from init level to the expected level.

Graphs



Simulation example: level control of a first-order, time delayed tank

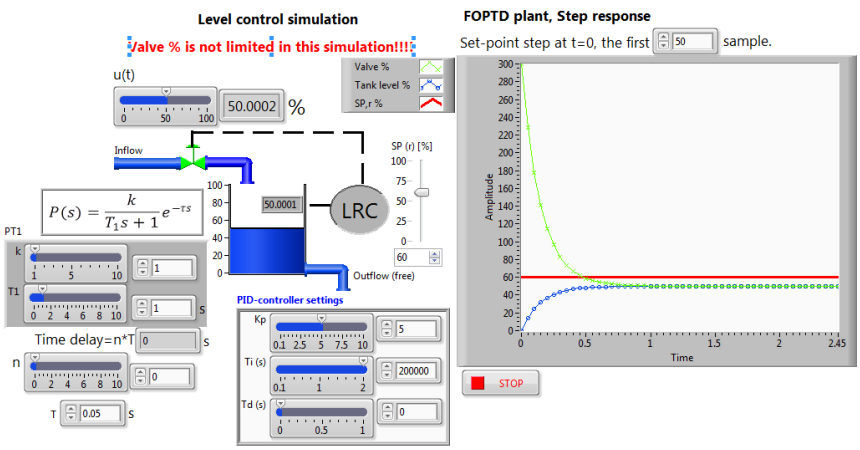


The saturation of the final control element

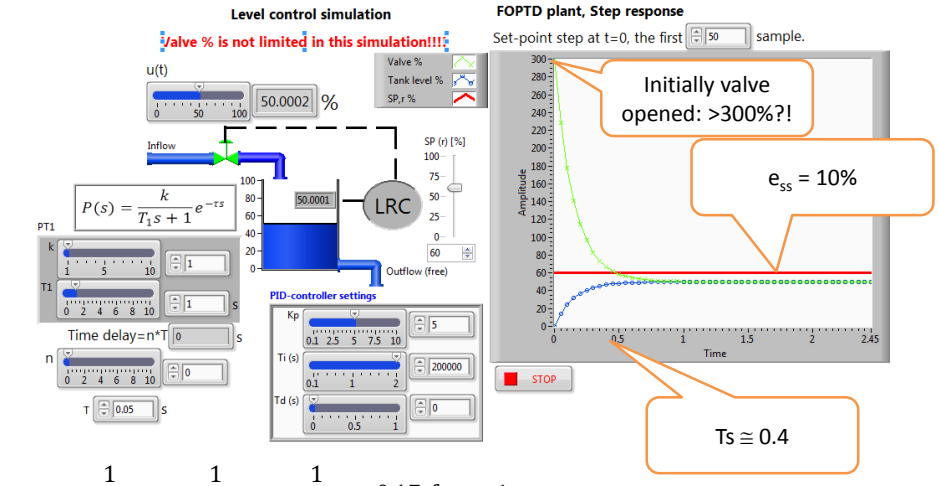
- Theoretically increasing controller gain decrease the steady-state error.
- Controller design process is based on simplifications which in the real word, over wide working zone is not valid.
- Strict linear system assumption in the practice not truth: nonlinear behaviors of the system component (hysteresis, saturation, dead-zone, etc.) are often unpredictable present.

Example: level control of a first-order tank

The actuator is the valve in the inlet pipe



No saturation is assumed

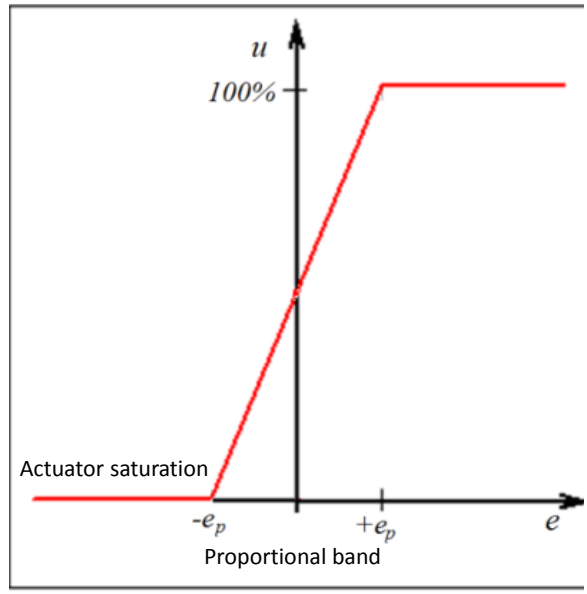


$$e_{ss} = \frac{1}{1 + k_0} = \frac{1}{1 + kK_P} = \frac{1}{1 + 5} = 0{,}17 \; for \; unit \; step$$

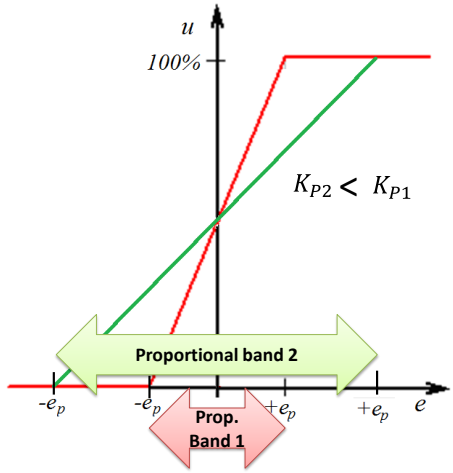
Reference changed: 0-60 $\rightarrow e_{ss}$ =0,17*60=10

Actuator saturation

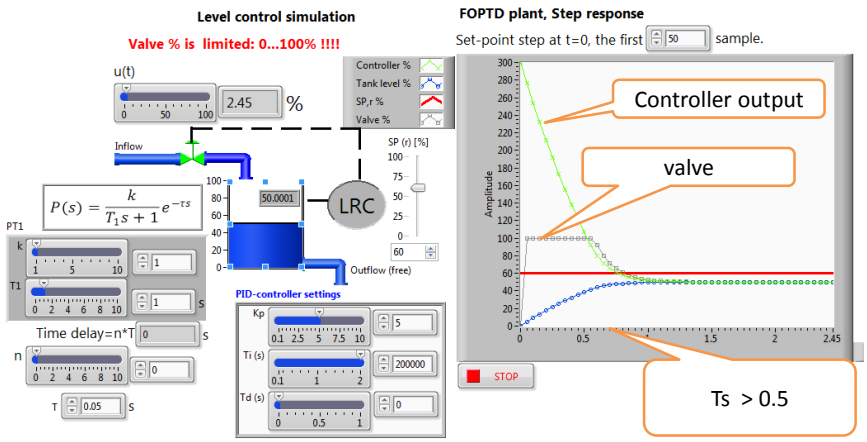
- The actuator is the valve in the inlet pipe and it can be opened between 0% and 100% .
- If the controller output is greatest than 100% or less than 0%, the valve “winds up”, this is the so called saturation.



The greater the controller gain, the narrower the proportional band



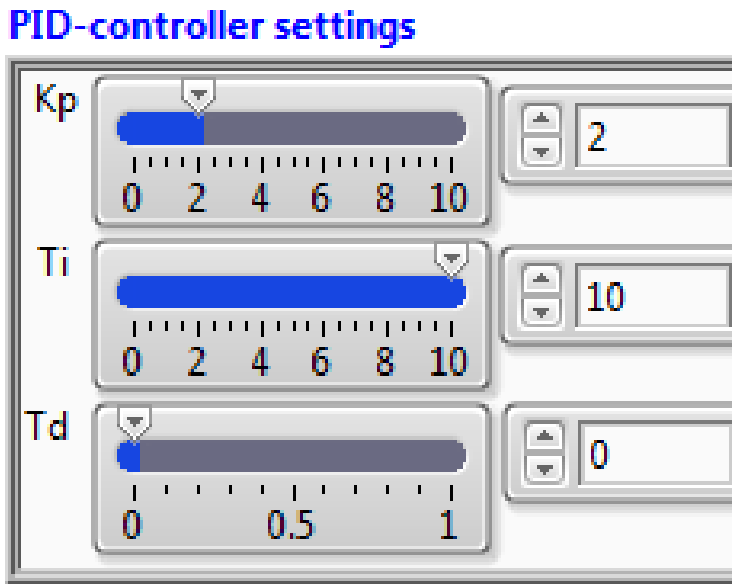
If Valve is limited, and the controller output is not, the performance degraded



The saturation is nonlinearity. The suggestions of classic control theory can be applied until the linear assumptions are valid.

Test

- Set the controller parameters
- Run the prongram
- See in the time diagrams the overshoot, the controller output, the valve and evaluate the Settling time



Test

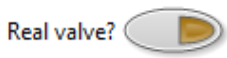
- Press real valve
- See in the time diagrams the overshoot, the controller output, the valve and evaluate the Settling time

Test

- Press anti wind-up too
- See in the time diagrams the overshoot, the controller output, the valve and evaluate the Settling time

In the following we consider linear system without limitations

- Switch off 

- Switch off 

THE PROPORTIONAL CONTROLLER: P-CONTROLLER

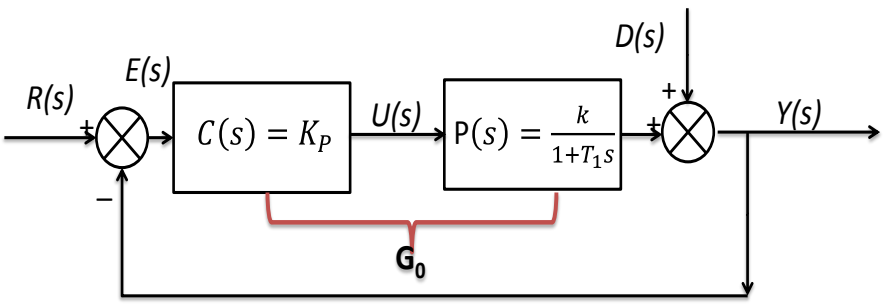
- The proportional controller transfer characteristic corresponds to the proportional (P) system.
- The control signal (*u*) (controller output) is always proportional to the error signal (*e*) (the input of the controller).

$$u(t) = K_P e(t)\ ,\ (u(0)=0)$$

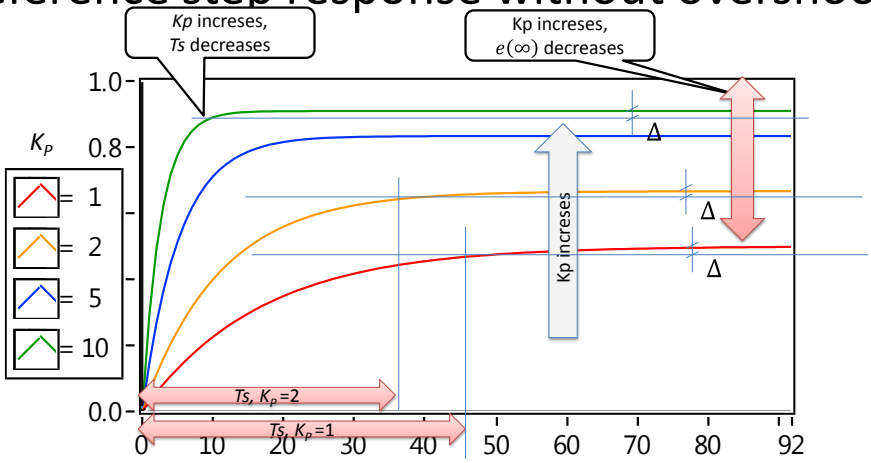
- *The transfer function of the P-controller*:

$$C(s) = K_P$$

P-CONTROLLER AND FIRST-ORDER PROCESS

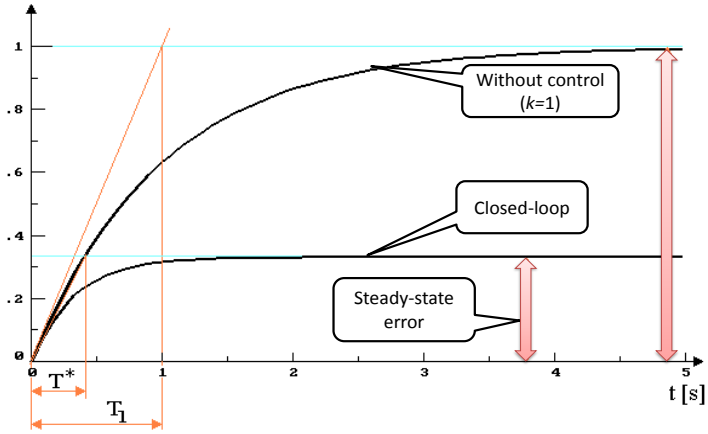


The first-order characteristic results closed-loop reference step response without overshoot.



With increasing K_P not only the steady-state error but the settling time could be reduced since with decreasing T_1^* the settling time decreases as well

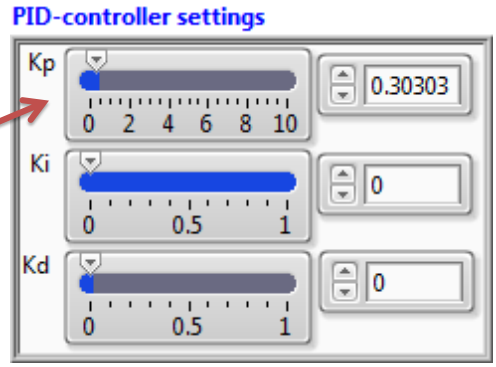
First-order plant and P-controller; response for unit step disturbance change, *r(t)*=0



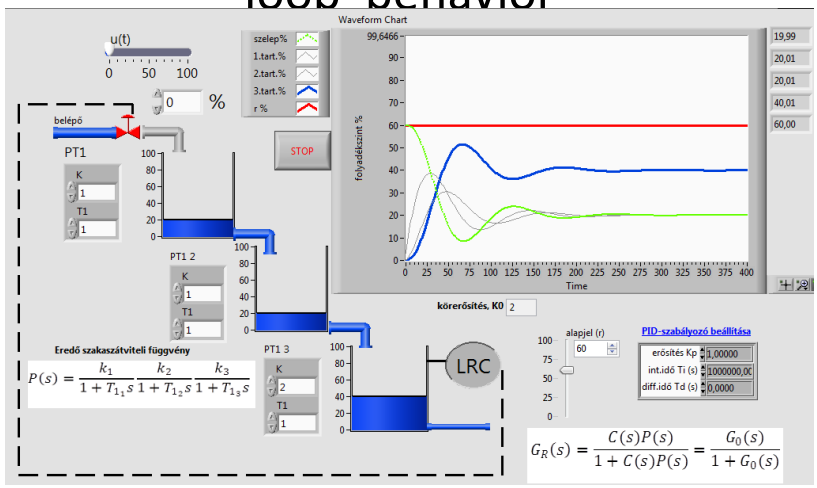
Test it!

PTH_time_PID_controller_MSc_GainPID.vi

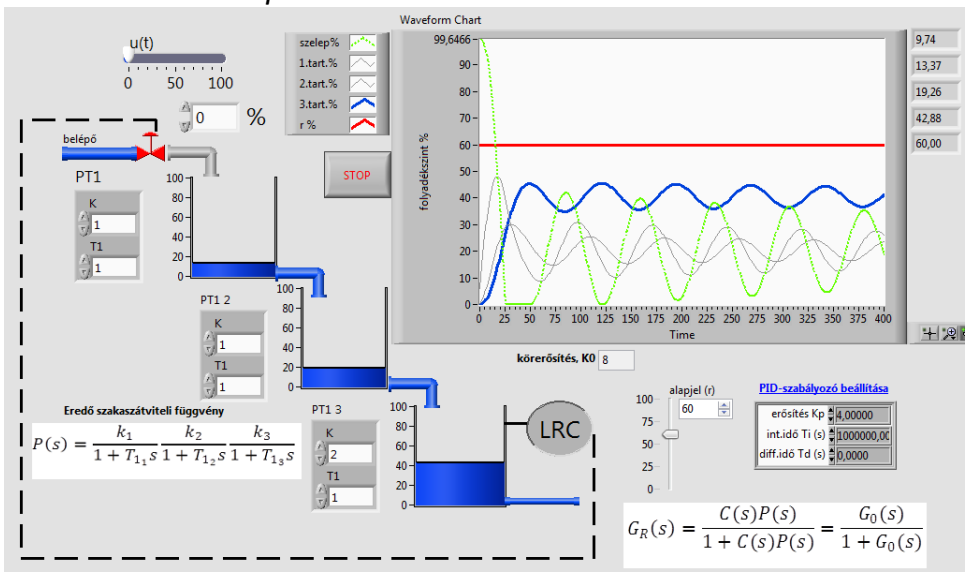
- Open the program
- Run
- Let Ki = K_d=0
- Change K_p with slider
- Explain, how affects the K_p change the performance and stability?



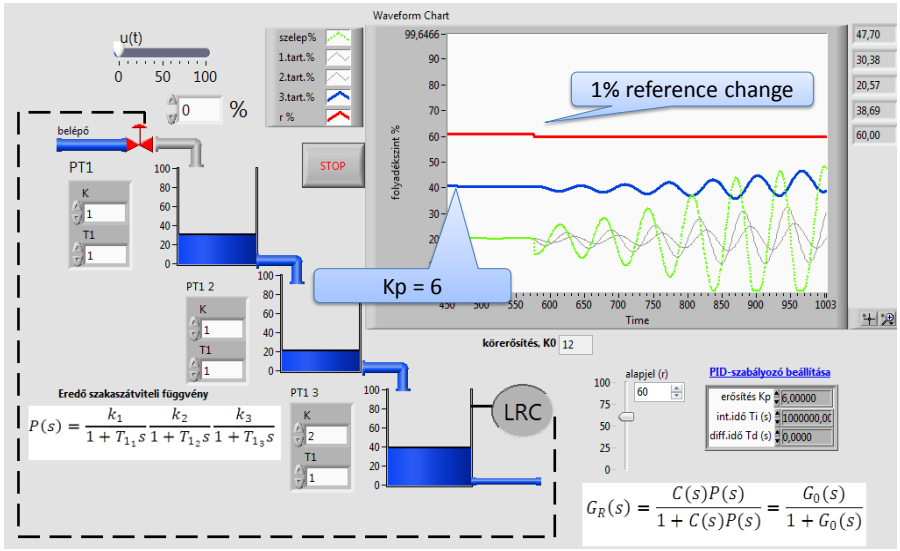
Third (or higher)-order system and P-controller: increasing Controller Gain results unstable closed-loop behavior



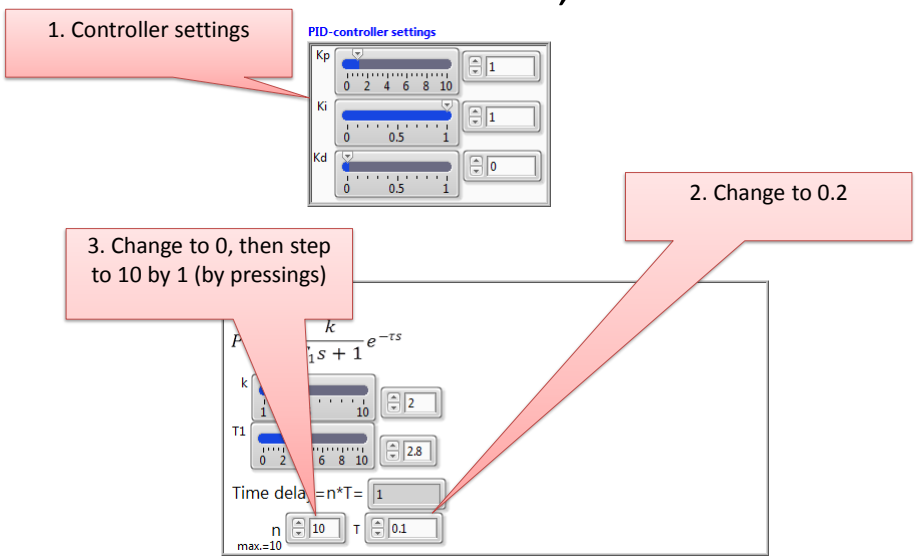
K_p=4, marginally stable



K_p=1 → 6, unstable control loop



Dead time can cause unstable closed-loop behavior as well, test it!



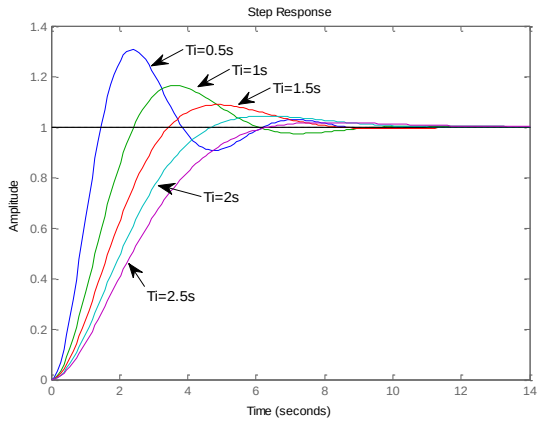
Integral controller (I-control)

- transfer characteristic corresponds to the integrator (I) system.
- I/O modell: $\frac{du(t)}{dt} = \frac{1}{T_i} e(t)$, $(u(0)=0)$.
- The controller will change the controller output signal, while the error signal is not zero
- The Integral controller transfer function:

$$C(s) = \frac{1}{T_i s} = \frac{k_i}{s}$$

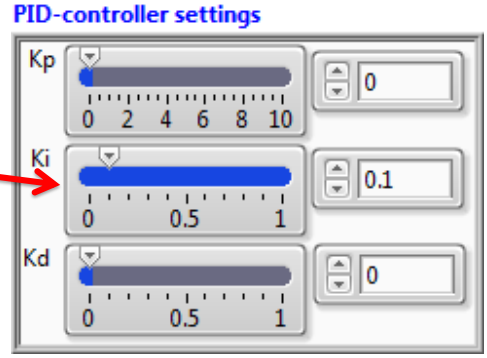
FIRST-ORDER PROCESS and I-CONTROLLER reference tracking

- $G_R(s) = \frac{1}{\frac{T_i}{k}T_1 s^2 + \frac{T_i}{k}s + 1}$.

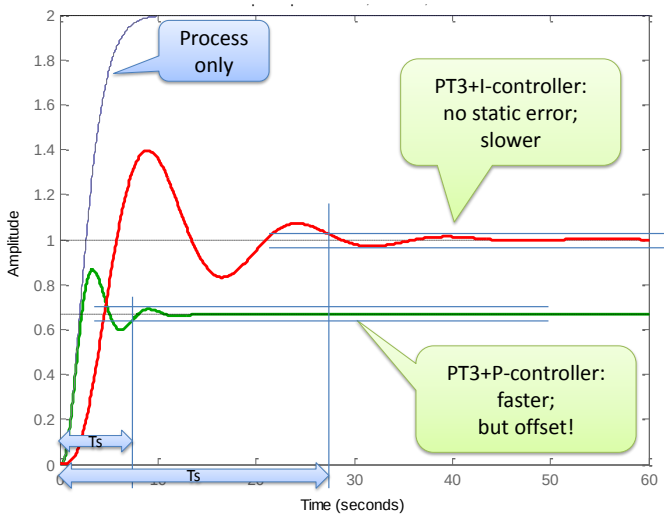


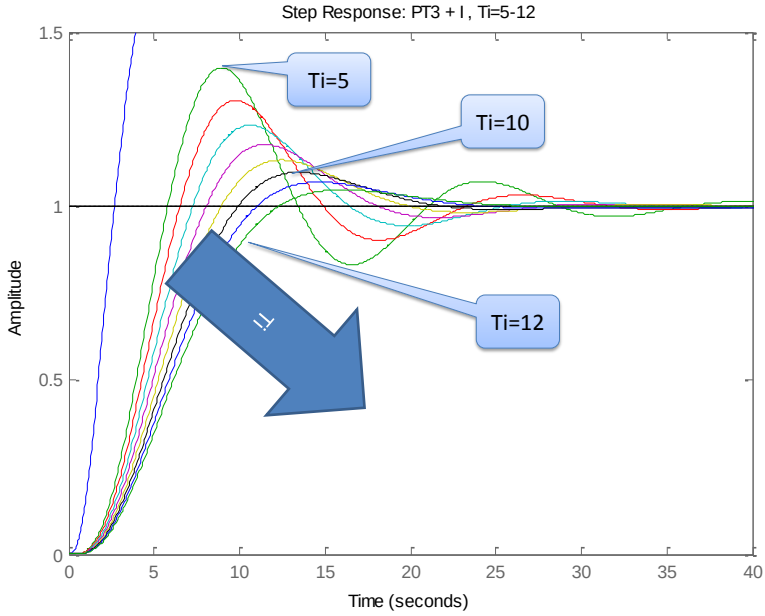
Test it!

- Open the program PTH_time_PID_controller_MSc_GainPID.vi
- Set K_p=0, K_d=0
- Run
- Change K_i with slider
- Increasing K_i = decreasing T_i
- How affects the K_i change the stability?



P- and I-controls, comparison in time domain





PI-controller

- We can incorporate the beneficial properties of the P- and I-controller into one PI-controller: the control will be faster and no steady-state error at step response

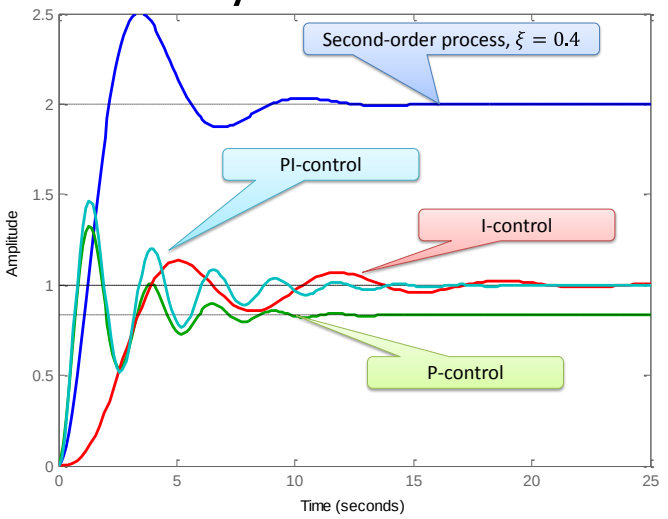
The I/O model of a PI-controller ($u(0)$=0):

$$u(t) = K_P e(t) + K_i \int_0^t e(\tau) d\tau$$

Or in Gain transfer form

$$u(t) = K_P e(t) + \frac{K_P}{T_i} \int_0^t e(t)\, dt$$

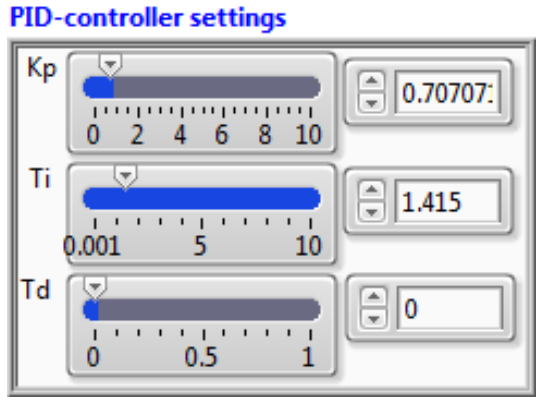
Second-order system with PI-controller



Test it!

PTH_time_PID_controller_MSc_TimePID_level.vi

- Open the program
- Set T_d=0
- Run
- Change K_p with slider
- Change T_i with slider
- How affect K_p and T_i the speed, overshoot and stability?

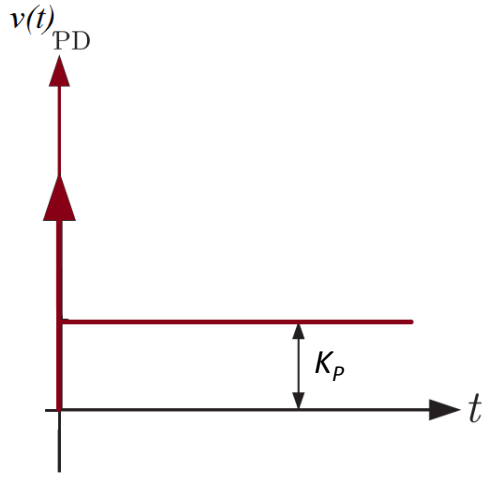


Proportional, derivative controller (PD-controller)

- I/O modell ($u(0)$=0):
- $u(t) = K_P\left(e(t) + T_D\frac{de(t)}{dt}\right)$
- ***Transfer function***:$C(s) = K_P + K_D s$ (D, P)
- or: $C(s) = K_P(1 + T_D s)$
- $K_D = K_P T_d$;

Step response

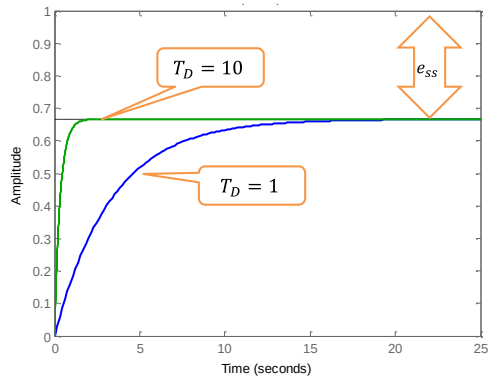
- The D-part acts only at the beginning of the response then the P-component works.
- The new steady-state depends only on the proportional component
- $v_{PD}(t) = K_D\delta(t) + K_P$
- (t>0)



Step response for reference change

Same steady-state errors

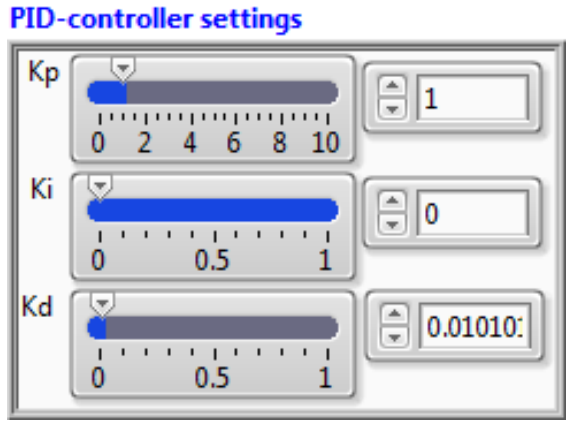
- Better, if the Larger time constant is cancelled
- Fast control with static error



Test it!

PTH_time_PID_controller_MSc_GainPID.vi

- Open the program
- Run
- Set K_p =1, K_i=0
- Change K_d with slider
- How affect K_d the speed, overshoot , steady-state error and stability?



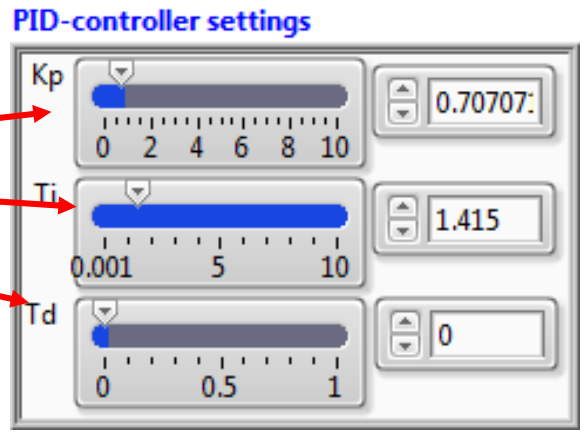
PID-controller

- The benefits of the PI and PD-controllers are merged in the PID-controller.
- The PID-controller ensures at least type-one control.
- It is the most popular controller type in the industry.

Test it!

PTH_time_PID_controller_MSc_TimePID_level.vi

- Open the program
- Run
- Change Kp with slider
- Change Ti with slider
- Change Td with slider
- Try to find a setting that results:
- No overshoot, no steady-state error and fast control (short settling time)



Intelligent control systems

TUNING OF A PID CONTROLLER

Tuning of PID-controller

- Controller tuning is the process of determining the controller parameters which produce the desired output.
- Controller tuning allows for optimization of a process and minimizes the error between the variable of the process and its set-point.
- The problem is twofold: the controller type and their parameters have to be set and tuned.

PID CONTROLLER TUNING IN TIME DOMAIN

The classic controller design steps

- Define the performance requirement of the control system
- Define the controlled process signal transfer characteristics, mostly by measurement of a step response
- Define the type and parameters of the PID controller
- Test the controller settings on a real system
- Fine tuning the controller

The effect of PID parameter changing on the characteristics of the control dynamics

Increasing parameter values:

PID parameters	Steady-state error	Overshoot	Settling time	Tendency to instability
K_P	Decrease	Increase	Decrease	Increase
T_i	Eliminate	Decrease	Increase	Decrease
T_D	Doesn't affect	Decrease	Decrease	Increase

Tuning in time domain

- Typical steps for refining the PID-controller settings are:
- Use *P* to decrease the rise time.
- Use *I* to eliminate the steady-state error.
- Use *D* to reduce the overshoot and settling time.

The classic PID-controller tuning methods

- *Closed-loop methods*, which are methods based on experiments on the already established closed-loop system.
- *Open-loop methods*, which are methods based on experiments on the open-loop system i.e. on the process itself, independent of the controller

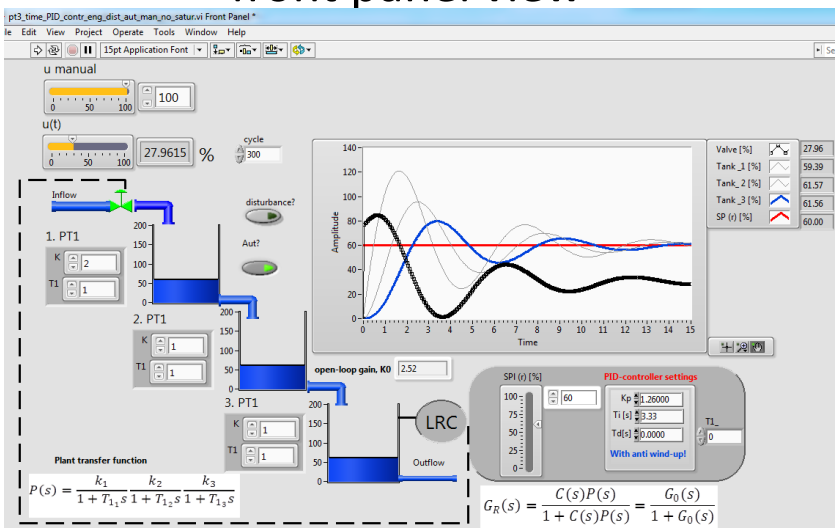
Closed-loop methods

- Trial and Error method
  - based on guess-and-check.
  - the proportional action is the main control, while the integral and derivative actions refine it.
  - Criterion: the control loop is established.

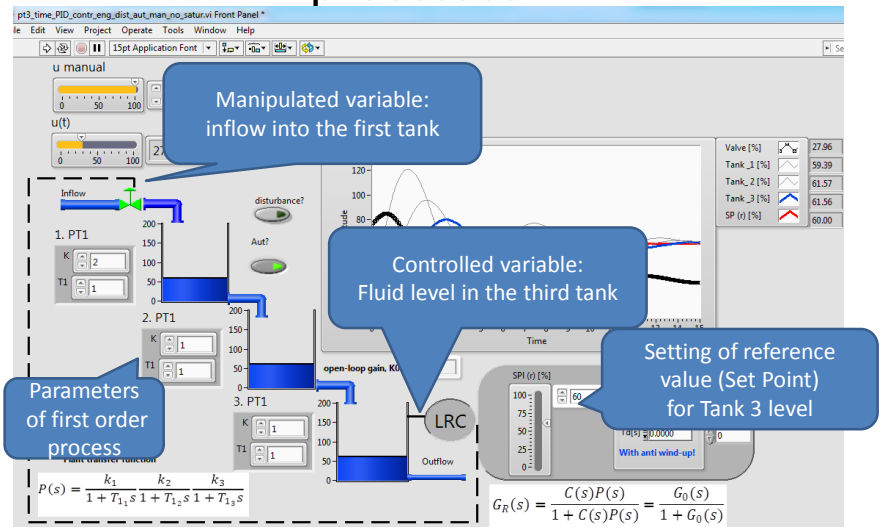
Trial and Error method

- The tuning procedure steps:
- The controller gain, K_P, is adjusted with the integral and derivative actions held at a minimum (Ki,Kd=0), until a desired output is obtained. If the control adequately fast, but not accurate enough, the next step:
- Find the proper integration time to remove all remaining regulatory differences.
- For a while let this setting to operate the system, then one have expertise on operation, and it seems necessary to further acceleration, the differentiator can be set.

Example : control of a third-order system, front panel view



Three, serial connected first order processes



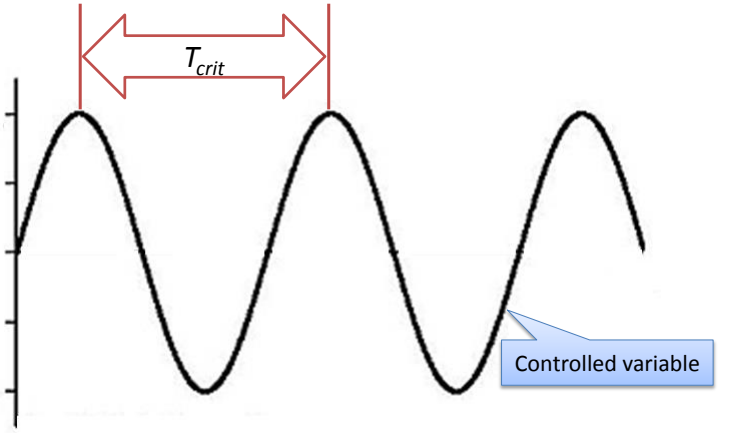
The Ziegler-Nichols' closed-loop tuning method

- also known as continuous cycling or ultimate gain method
  - Criterion: the control loop is established.
- The ultimate or critical proportional gain K_{Pcrit} of a P-controller must be found, and the ultimate (or critical) period T_{crit} of the sustained oscillations is measured.
- The critical proportional gain is the gain which causes sustained oscillations in the output signals in the control system without the control signal reaching the maximum or minimum limits.

Tuning procedure

- At the controller, select proportional-only (P-ONLY) control:
  - set K_P to the lowest value ; T_i to infinity ; T_D to zero. (or Ki=Kd =0)
- 2. Adjust the controlled system manually to the desired operating point (start-up control loop).
- 3. Set the output of the controller to the manually adjusted value and switch to automatic operating mode.
- 4. Gradually increase K_P until the controlled variable encounters harmonic oscillation. If possible, small step changes in the set-point should be made during the K_P adjustment to cause the control loop to oscillate.
- 5. Take down the adjusted K_P value as critical proportional-action coefficient K_{Pcrit}.
- 6. Determine the time span for one full oscillation amplitude as T_{crit}

Determine the time span for one full oscillation amplitude



PID parameter can be calculated from the following table

Controller type	K_P	T_i	T_D
P-	$0.5K_{Pcrit}$		-
PI-	$0.45K_{Pcrit}$	$0.83T_{crit}$	-
PID-	$0.6K_{Pcrit}$	$0.5T_{crit}$	$0.125T_{crit}$

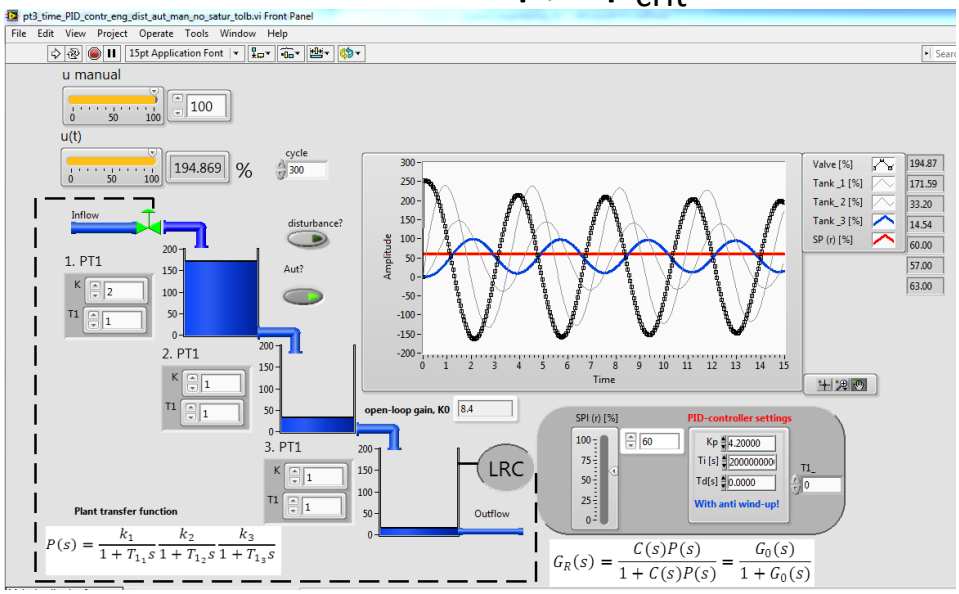
The quarter amplitude decay ratio

- The main design criterion was to obtain good rejection of load disturbances specified as quarter amplitude decay ratio.
- The quarter amplitude decay ratio gives systems with very poor damping.
- Settings was modified that gives better damped system response.

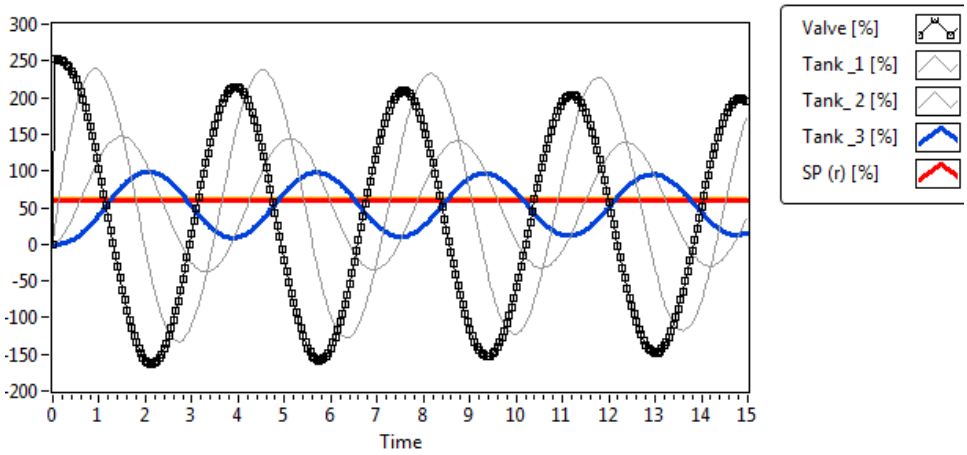
Modified Z-N closed-loop tuning

Controller type	K_P	T_i	T_D
PID-original	$0.6K_{Pcrit}$	$0.5T_{crit}$	$0.125T_{crit}$
PID-overshoot	$0.33K_{Pcrit}$	$0.5T_{crit}$	$0.33T_{crit}$
PID- no overshoot	$0.2K_{Pcrit}$	$0.3T_{crit}$	$0.5T_{crit}$

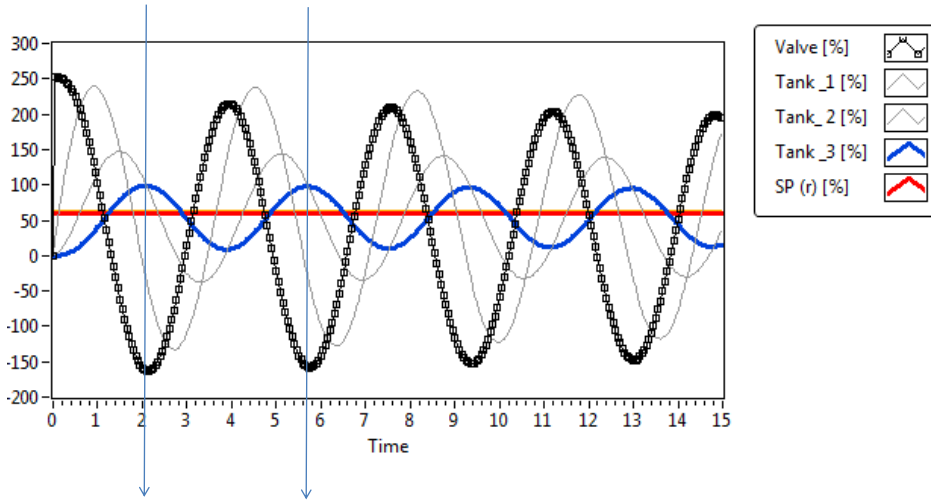
Z-N closed loop, Kp_{crit}=4.2



Exported image



Exported image:Tcrit = 5.7-2.0=3.7s



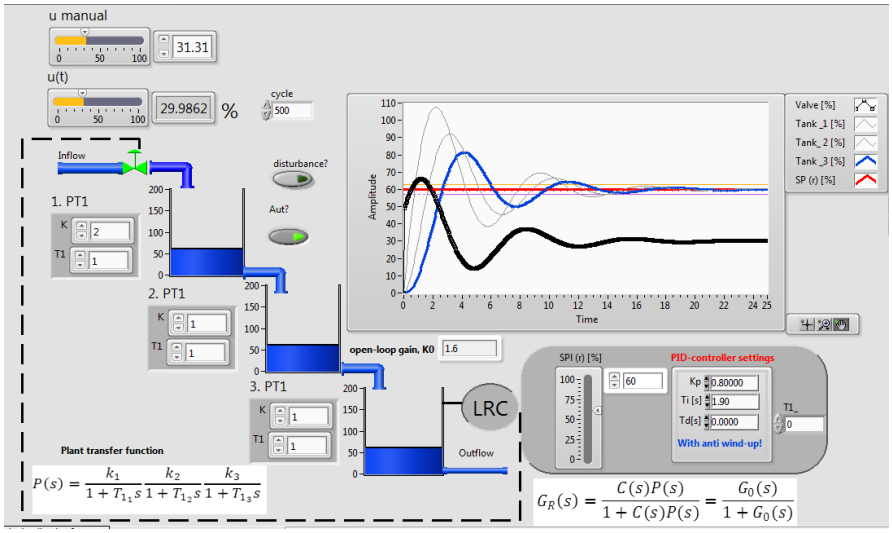
Modified Z-N closed-loop tuning

Controller type	K_P	T_i	T_D
PID-original	$0.6K_{Pcrit}$	$0.5T_{crit}$	$0.125T_{crit}$
PID-overshoot	$0.33K_{Pcrit}$	$0.5T_{crit}$	$0.33T_{crit}$
PID- no overshoot	$0.2K_{Pcrit}$	$0.3T_{crit}$	$0.5T_{crit}$

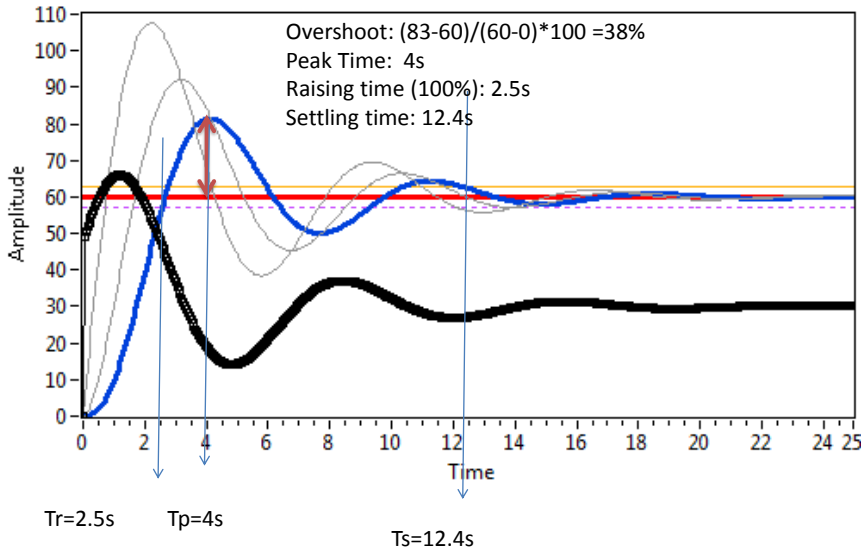
Kp,crit=4.2
Tcrit=3.7s

Kp=0.2*4.2 ≅ 0.8 Ti=0.5*3.7s≅1.9s

Testing ….



Dynamic performance test



Advantages of Z-N closed-loop tuning

- No a priori information on process required
- Applicable to all stable processes
- Only a single experimental test is needed
- The controller settings are easily calculated

Disadvantages of Z-N closed-loop tuning

- Time consuming
- Loss of product quality and productivity during the tests
- Continuous cycling may cause the violation of process limitation and safety hazards
- Not applicable to open-loop unstable process
- First-order and second-order process without time delay will not oscillate even with very large controller gain. This fact motivates the Relay Feedback Method.
- *Relay Feedback Method* forces the system to oscillate by a relay – or on/off – controller.

Open-loop PID tuning methods
Reaction curve methods

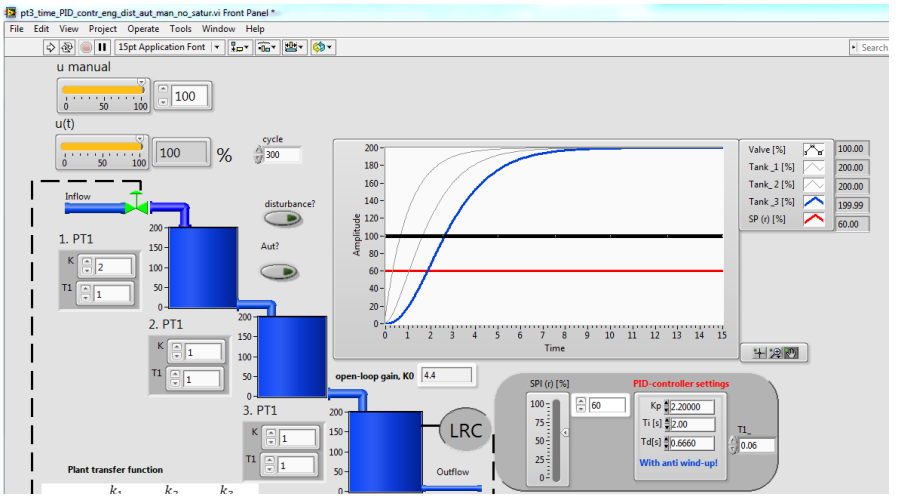
- It applies to plants whose unit-step response resembles an S-shaped curve with no overshoot. This S-shaped curve is often called as the reaction curve.
- it is assumed that the process transfer function *P(s)*, can be approximated with FOPTD
- $P(s) = \frac{k}{T_1 s+1} e^{-\tau s}$
- These controller parameters will typically give a response with an overshoot about 20-25% and good settling time.

Steps of tuning procedure

- Make an open-loop step test.
- From the process reaction curve determine the FOPDT parameters: k, T_1, τ
- $P(s) = \frac{k}{T_1 s+1} e^{-\tau s}$

- Determine the loop tuning constants using the table

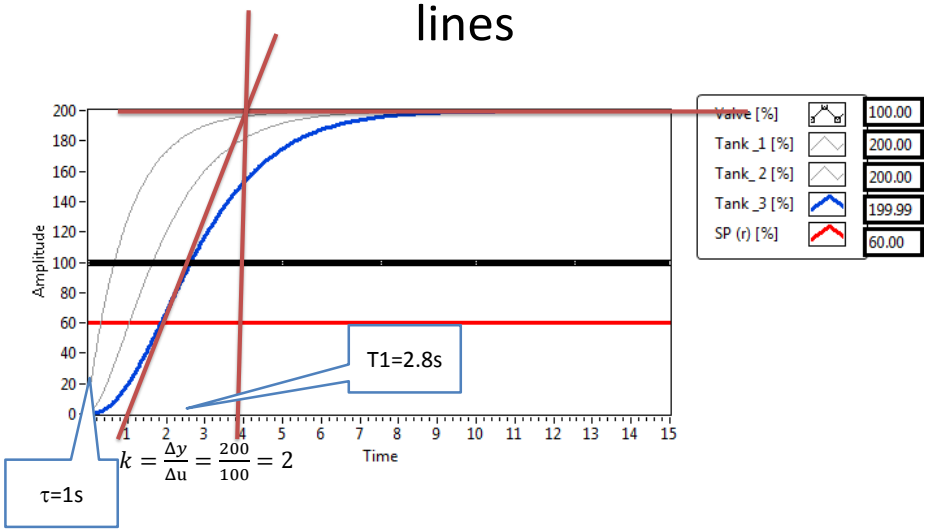
Example : PI-control of a third-order process
1. Step respons measurement



Step respons measurement

- Measure the third-order system step response and approximate wit a FOPDT modell
- Set the controller to manual mode
- Change the valve position from 0% to 100%
- Export the image

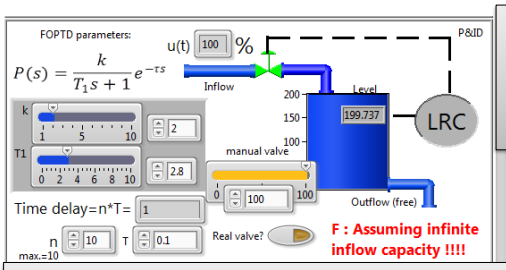
Step respons measurement and graphic approximations of system parameters with tangent lines



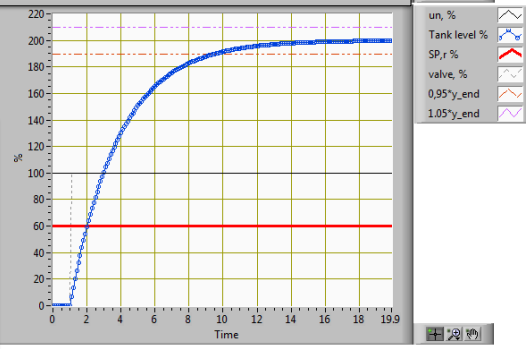
The FOPDT modell

- $P(s) = \frac{k}{T_1 s+1} e^{-\tau s}$
- $P(s) = \frac{2}{2.8s+1} e^{-s}$
- (Instead of $P(s) = \frac{2}{(s+1)^3}$!)

Test the step response



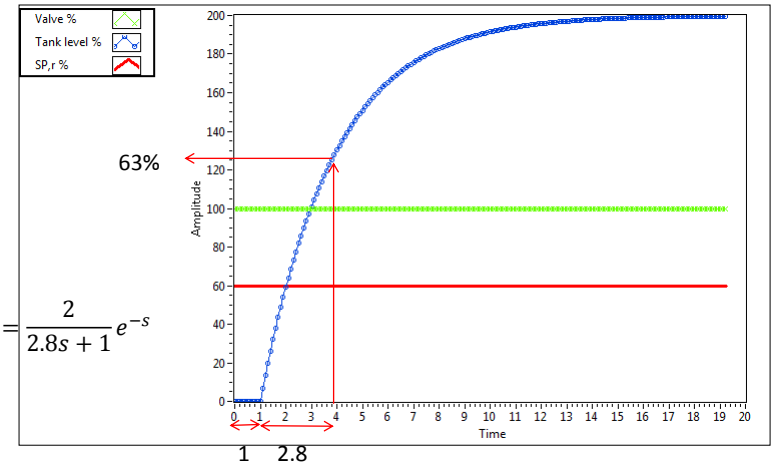
$$P(s) = \frac{2}{2.8s + 1} e^{-s}$$



Set the process parameters (k, T1 and delay time)
Controller in manual mode
Open valve manually to 100%
Run the program

Step response of the FOPDT modell

$$P(s) = \frac{2}{2.8s + 1} e^{-s}$$



Ziegler-Nichols open-loop tuning method

Controller type	K_P	T_i	T_D
P-	$\frac{1}{k}\frac{T_1}{\tau}$		
PI-	$\frac{0,9}{k}\frac{T_1}{\tau}$	$3,33\tau$	
PID-	$\frac{1,2}{k}\frac{T_1}{\tau}$	2τ	$0,5\tau$

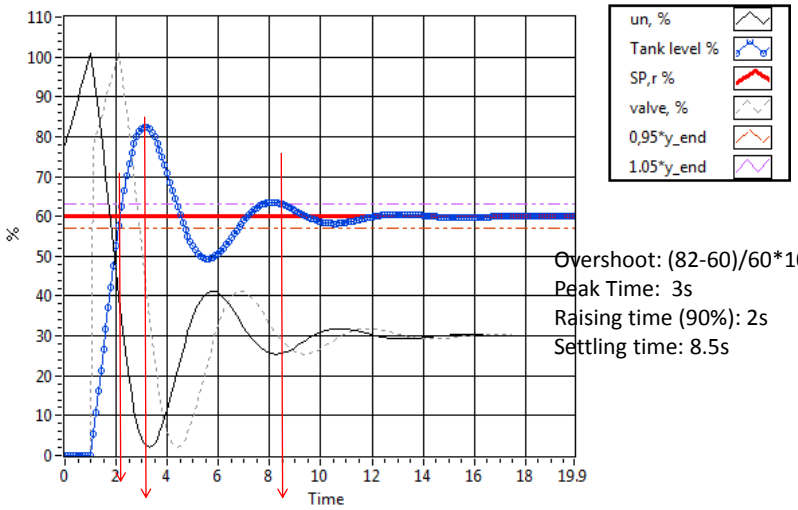
Ziegler-Nichols open-loop tuning method

τ=1s T1=2.8s k=2

Kp=0.9*2.8/2/1=1.26
Ti=3.33*1s=3.33s

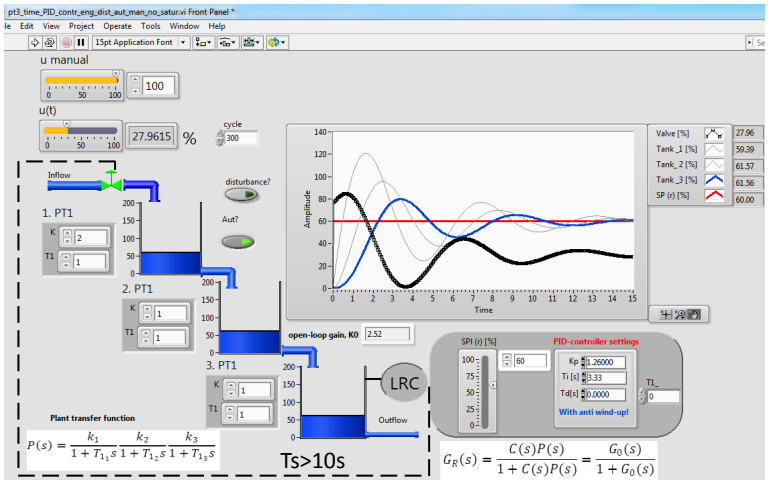
Controller type	K_P	T_i	T_D
P-	$\frac{1}{k}\frac{T_1}{\tau}$		
PI-	$\frac{0,9}{k}\frac{T_1}{\tau}$	$3,33\tau$	
PID-	$\frac{1,2}{k}\frac{T_1}{\tau}$	2τ	$0,5\tau$

Test on FOPDT model control

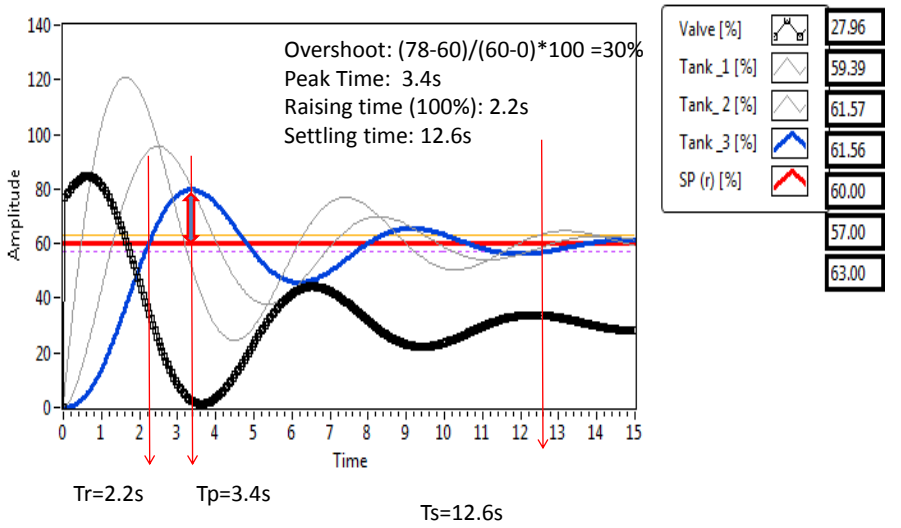


Overshoot: (82-60)/60*100=36%
Peak Time: 3s
Raising time (90%): 2s
Settling time: 8.5s

Test on Third-order system



Result of the Ziegler-Nichols open-loop tuning



Ziegler-Nichols open-loop tuning method

- Advantages
- Quick and easier to use than other methods
- It is a robust and popular method
- Disadvantages
- Approximations for the K_P , T_i and T_D values might not be entirely accurate for different systems
- It does not hold for I, D and PD-controllers.

CHEN–HRONES–RESWICH (C–H–R) method

- Modified Z-N open loop tuning method
- The method use “quickest response without overshoot” or “quickest response with 20% overshoot” as design criterion.
- The method takes into account also that tuning for set-point responses and load disturbance responses are different.

C-H-R, load rejection

Controller type	No overshoot			20% overshoot		
	K_P	T_i	T_D	K_P	T_i	T_D
P-	$\frac{0.3}{k}\frac{T_1}{\tau}$			$\frac{0.7}{k}\frac{T_1}{\tau}$		
PI-	$\frac{0.6}{k}\frac{T_1}{\tau}$	4τ		$\frac{0.7}{k}\frac{T_1}{\tau}$	2.3τ	
PID-	$\frac{0.95}{k}\frac{T_1}{\tau}$	2.4τ	0.42τ	$\frac{1.2}{k}\frac{T_1}{\tau}$	2τ	0.42τ

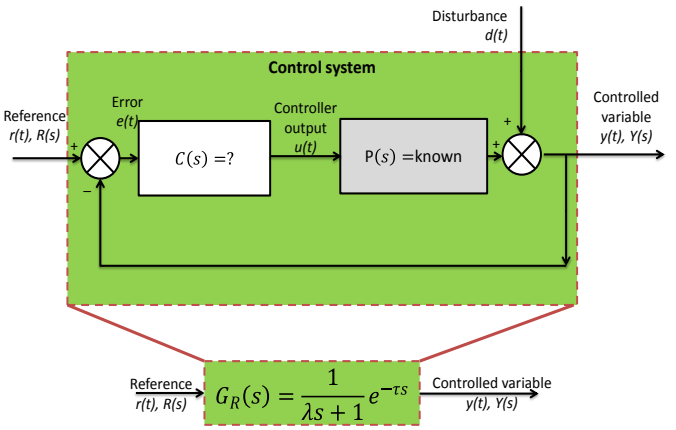
C-H-R, reference tracking

Controller type	No overshoot			20% overshoot		
	K_P	T_i	T_D	K_P	T_i	T_D
P-	$\frac{0.3}{k}\frac{T_1}{\tau}$			$\frac{0.7}{k}\frac{T_1}{\tau}$		
PI-	$\frac{0.35}{k}\frac{T_1}{\tau}$	1.2τ		$\frac{0.6}{k}\frac{T_1}{\tau}$	τ	
PID-	$\frac{0.6}{k}\frac{T_1}{\tau}$	τ	0.5τ	$\frac{0.95}{k}\frac{T_1}{\tau}$	1.4τ	0.4τ

Model based controls which result PID-control law

- *Internal Model Principle and Internal Model Control*
- Internal Model Principle states that control can be achieved only if the control system encapsulates, either implicitly or explicitly, some representation of the process to be controlled.
- Instead of fixing a control structure and then attempting to “extract” optimality from this controller , IMC postulate a model, state desirable control objectives, and, from these, proceed in a straightforward manner to obtain both the appropriate controller structure and parameters.
- It can be stated, that depending on the numerical aproximation of the dead time , the IMC controller for FOPDT process is PI- or PID-controller.

LAMBDA TUNING method



- assumes a first order time delayed closed-loop dynamic by unity gain
- $G_R(s) = \frac{1}{\lambda s+1} e^{-\tau s}$
- λ is the tuning parameter
- Known: $P(s)$;
- Search: $C(s)$).

LAMBDA TUNING method

- the control variable follows the step reference change without overshoot and without steady-state error.
- This *overdamping* feature can be especially useful in applications where the process variable must be maintained near some limiting value that the process variable must not cross.
- Smaller λ values increase the controller performance but take it closer to the instability region.

LAMBDA TUNING method

Controller type	K_P	T_i	T_D	Recommended λ/τ ($\lambda > 0.2\tau$ always)
PI-	$\frac{1}{k}\frac{T_1}{\lambda}$	T_1	-	>0.25
Improved PI-	$\frac{1}{k}\frac{2T_1+\tau}{2\lambda}$	$T_1+\frac{\tau}{2}$	-	>1.7
PID-	$\frac{1}{k}\frac{2T_1+\tau}{2(\lambda+\tau)}$	$T_1+\frac{\tau}{2}$	$\frac{\lambda\tau}{2(\lambda+\tau)}$	>1.7

The lambda tuning parameters for a PID-controller for first-order time delayed process with parameters gain: k, time constant: T_1 and time delay: τ:

Only the proportional gain needs to be adjusted!

The integral time is simply set equal to the process time constant.

LAMBDA TUNING method

- The proportional gain is inversely related to λ.
- If λ is small i.e. the closed-loop is “fast”, the controller gain must be large.
- Similarly, if λ is large (closed-loop is “slow”) the c
- Lambda tuning tends to make a slow process even slower, causing the process variable to remain out of steady state for a long time.
- λ is generally assigned a value between T_1 and $3T_1$, making the closed-loop response to a set-point change up to three times longer than the corresponding open-loop step response.

Lambda tuning let λ=3 (>2.8=T1)

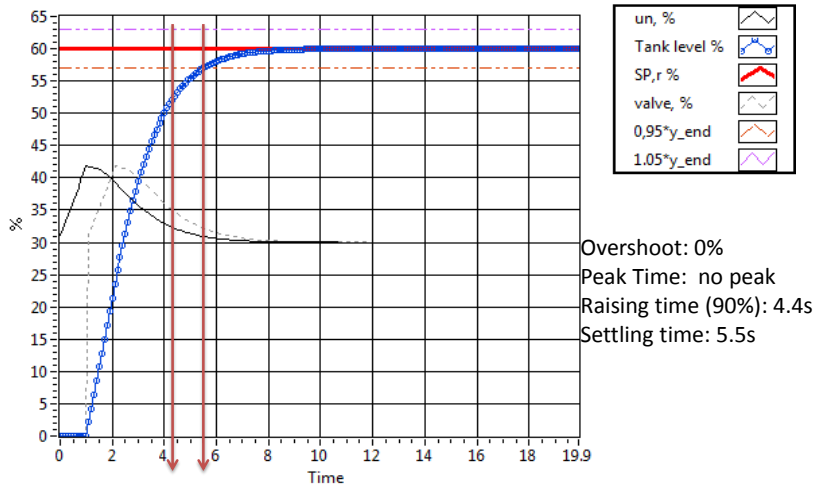
τ=1s T1=2.8s k=2

Szabályozó típusa	K_P	T_i	T_D	Ajánlás: λ/τ $(\lambda > 0.2\tau)$
PI-	$\frac{1}{k}\frac{T_1}{\lambda}$	T_1	-	>0.25
Improved PI-	$\frac{1}{k}\frac{2T_1+\tau}{2\lambda}$	$T_1+\frac{\tau}{2}$	-	>1.7
PID-	$\frac{1}{k}\frac{2T_1+\tau}{2(\lambda+\tau)}$	$T_1+\frac{\tau}{2}$	$\frac{\lambda\tau}{2(\lambda+\tau)}$	>1.7

$T_1 < \lambda < 3T_1$ $\lambda > \tau$ lower limit

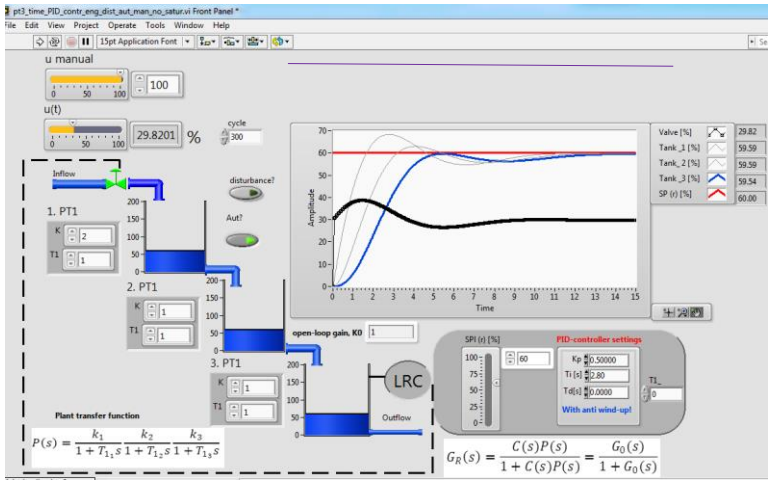
Kp=2.8/2/3≅0.5
Ti=2,8s

Test on FOPDT model control



Overshoot: 0%
Peak Time: no peak
Raising time (90%): 4.4s
Settling time: 5.5s

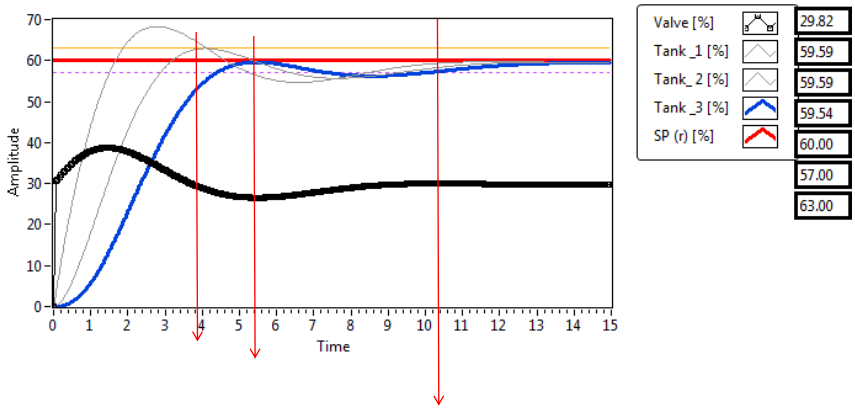
Test on Third-order system



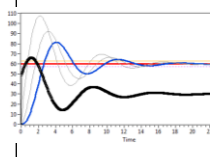
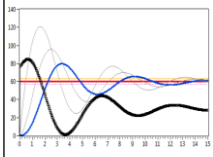
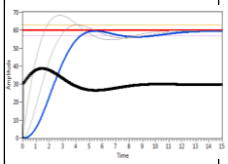
Ts=10s

Lambda tuning result, λ=3

Overshoot: 0%
Peak Time: 5.4s
Raising time (90%): 3.7s
Settling time: 10.5s



Performance comparison

	Z-N closed-loop, modified	Z-N open-loop	Lambda =3
Kp	0.8	1.26	0.5
Ti, s	1.9	3.33	2.8
Overshoot, %	38	30	0
Raising time(100%), s	2.5	2.2	3.7 (90%)
Peak Time, s	4	3.4	(5.4)
Settling time, s	12.4	12.6	10.5
			

Intelligent control systems

MODEL-BASED CONTROLS

Optimum system

- A control system is optimum when the elected performance index is minimized.
- The optimum value of the parameters depends directly upon the definition of optimum, that is, the performance index
- General form of the performance integral

$$I = \int_0^T f(e(t), r(t), y(t), t)dt$$

Control error dependent Performance integrals

$$ISE = \int_0^T e^2(t)dt$$

$$IAE = \int_0^T |e(t)|\,dt$$

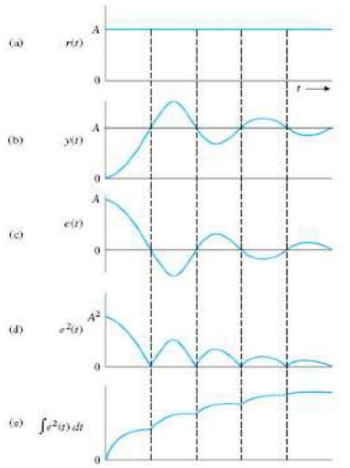
$$ITAE = \int_0^T t\,|e(t)|\,dt$$

$$ITSE = \int_0^T te^2(t)dt$$

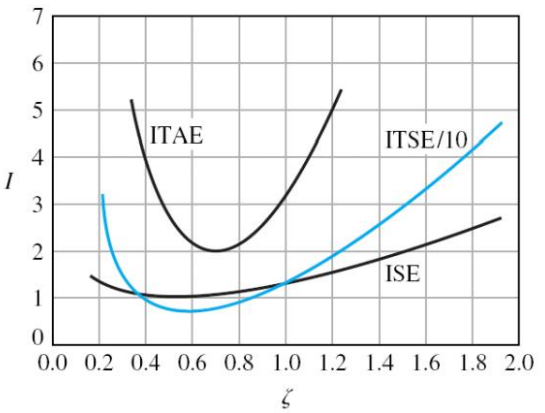
- ISE - Integral Squared Error
- ITAE - Integral of Time Multiplied by Absolute Error
- IAE - Integral of the Absolute Magnitude of the Error

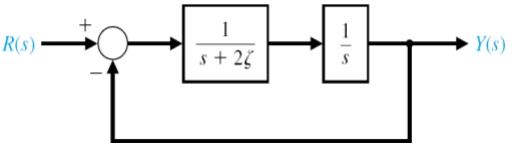
ISE - Integral Squared Error

$$I_1 = \int_0^T e^2(t)dt$$

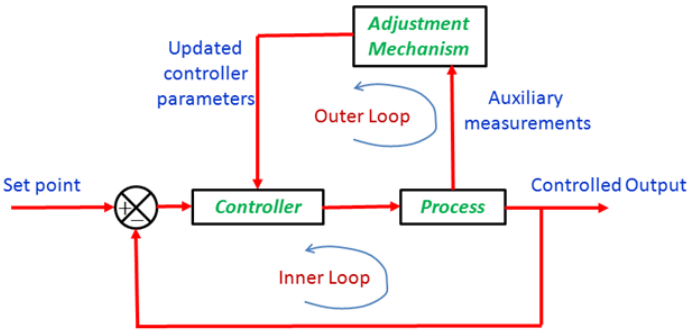


Comparison of the integrals





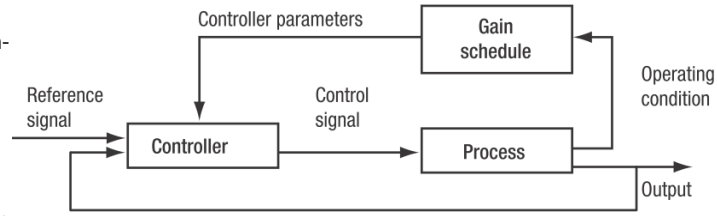
Adaptive control



- If the controller must adapt to a controlled system with parameters which vary, or are initially uncertain.

Gain scheduling

Conceptual diagram of gain-scheduling control



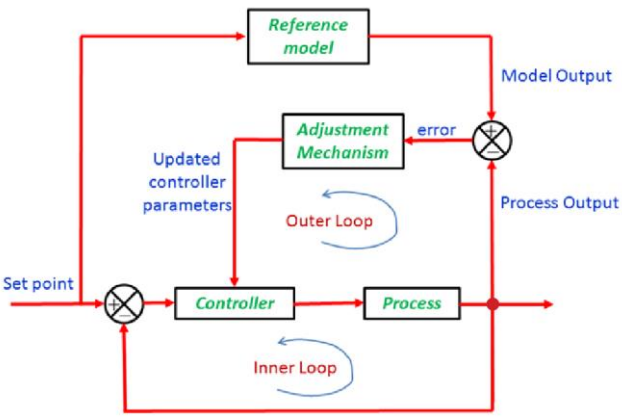
- Gain scheduling based on measurements of operating conditions of the process to compensate for variations in process parameters or known nonlinearities of the process
- the parameters are changed in an open- loop or pre- programmed manner
- Gain scheduling is a very useful technique for reducing the effects of parameter variations.
- It is customary to keep the overall gain constant. In case of changes in the process (or valve characteristics or measuring element), *controller gain* should be tuned in such a manner that overall gain remains constant.

Self-adaptive control

- When the process is poorly known the *self-adaptive control* may be helpful.
- The main reason for using an adaptive controller is that the process or its environment is changing continuously.
- To simplify the problem, it can be assumed that the process has constant but unknown parameters.
- The term self-tuning was used to express the property that the controller parameters converge to the controller that was designed if the process was known.
- A self adaptive controller optimizes the value of certain objective function (criterion) in order to obtain updated controller parameters.
- Two examples of self adaptive controllers are Model Reference Adaptive Control (MRAC) and Self-Tuning Regulator (STR)

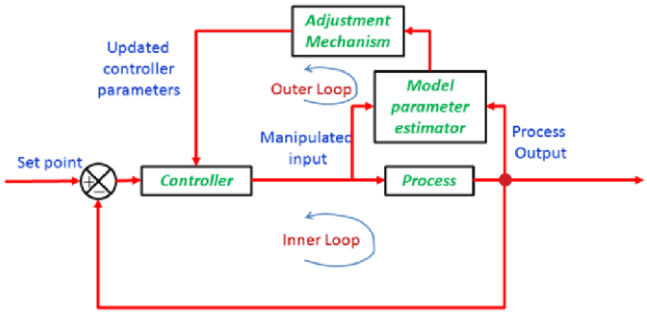
Model Reference Adaptive Control

- The inner loop contains the regular feedback mechanism
- The outer loop contains an ideal reference model which the process needs to follow.
- The process and model outputs are compared and the error function is minimized through a suitable optimization routine in order to arrive at the re-tuned controlle parameters.

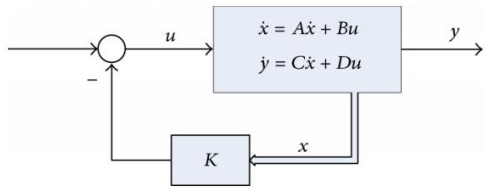


Self-Tuning Regulator

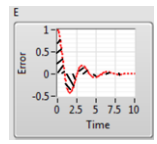
- Self-Tuning Regulator estimates the model parameters by measuring process inputs and outputs.
- The re-tuned model eventually guides the controller parameter adjustment mechanism.



Optimal control



- Linear–Quadratic–Regulator (LQR) control
- The settings of a (regulating) controller governing either a machine or process (like an airplane or chemical reactor) are found by using a mathematical algorithm that minimizes a cost function with weighting factors supplied by a human (engineer).
- The cost function is often defined as a sum of the deviations of key measurements, from their desired values.
- The algorithm thus finds those controller settings that minimize undesired deviations.
- The magnitude of the control action itself may also be included in the cost function.



Model-based controls

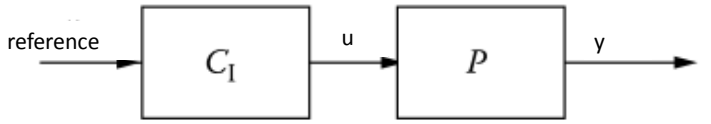
- In model-based control (MBC) , computers use a process model to make control decisions.
- By comparison to PID-type approaches, the MBC action is “intelligent” and has shown benefits in uniformity, disturbance rejection, and setpoint tracking, all of which translate into better process safety and economics.
- On the other hand, MBC requires higher skill by the operator and engineer, and computational power.
- higher level of understanding of process models means better process understanding, better personnel training, diagnosis ability, and process management.
- There are many successful commercial controllers, and some distributed control system (DCS) suppliers offer MBC software packages with their equipment.

THE STRUCTURE OF MBC

- The controller does not have PID components and, in general, there is only one tuning parameter per controlled variable—the choice of how fast the controlled variable should move to the setpoint.

Feedforward MBC

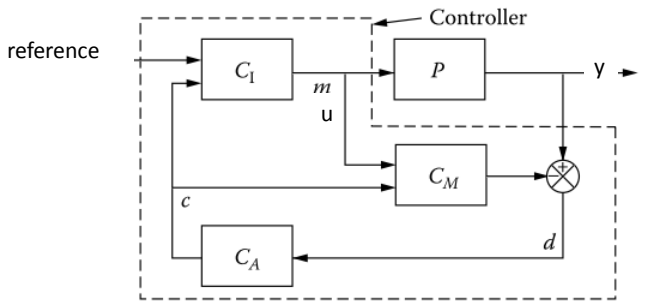
- If the model is perfect and no constraints exist, the open loop structure will work.
- C_I represents the model inverse: MBC determines the process input that causes a desired process output response



C_I = Inverse controller model of the process
P = Process

Feedback MBC

- nearly all controller models are not perfect; therefore, they require some form of feedback correction.
- The difference between the model (C_M) and the process output (y) is monitored and used to adjust (C_A) a controller feedback, which usually is either a bias to the setpoint or a model coefficient. The controller consists of the three functions C_I, C_M, and C_A, enclosed by the dashed line.



Intelligent control decisions

- The common aim of this strategy is to make intelligent control decisions
- The choices of model structure, control objective, adjusting mechanism, and adjustable parameters have led to many variations in model-based controllers
- MBCs are generally classified by model type.
  - The choice of model structure affects implementation, limits choices of other controller functions, and sets the accuracy of control decisions.
  - Choices for control objective, adjustment mechanisms, and adjustable parameters, and methods for other features such as constraint handling, optimization, and data reconciliation are influential but secondary.
- ***In all cases, control is only as good as the model is a true representation of the process***. Initializing the controller with a model that has been validated by process testing is the first and most critical implementation step.

Advantage/Disadvantage of IMC

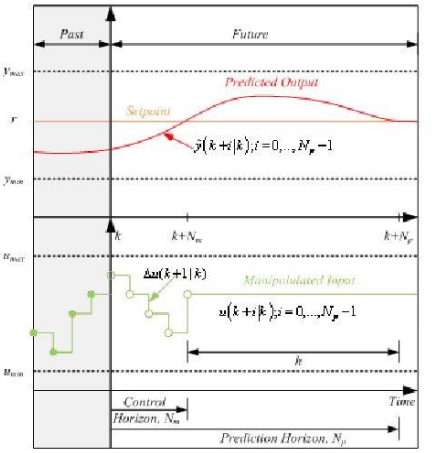
- One technical drawback to IMC is that the model is linear and stationary
- If either the process gain or the time cconstants change, the model-based calculations become either too aggressive or sluggish, and therefore, retuning is required.
- This is not different from PID control, in which changes in process gain or dynamics also require retuning.
- An advantage is that SISO IMC can be implemented with the skill of most control engineers using standard DCS or PLC functions.
- As a note, lambda tuning of PID controllers is based on the IMC approach.

MIMO model

- Process and controller transfer characteristic descriptions by: State-space representation
  - The *state-space representation* ("time-domain approach") provides a compact way to model and analyze systems with multiple inputs and outputs.

Model Predictive Control (MPC)

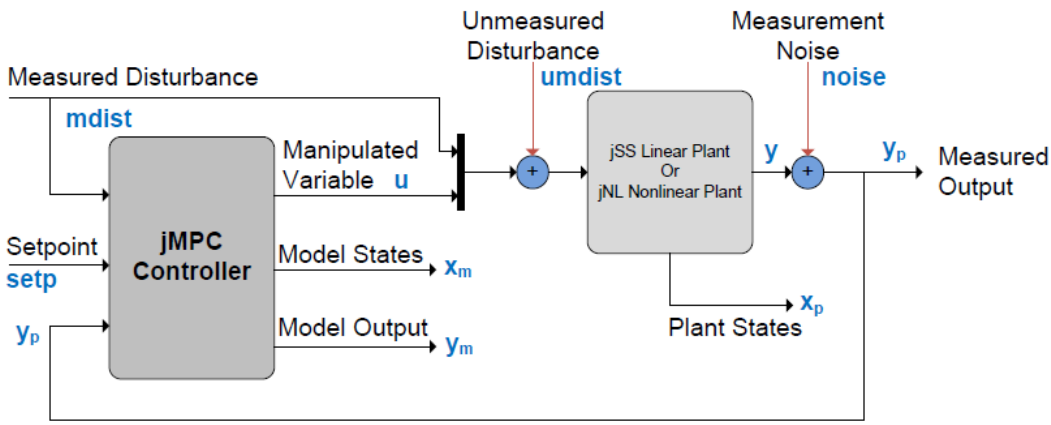
- The basic idea of MPC is to predict the future behavior of the controlled system over a finite time horizon and compute an optimal control input that, while ensuring satisfaction of given system constraints, minimizes an a priori defined cost functional.



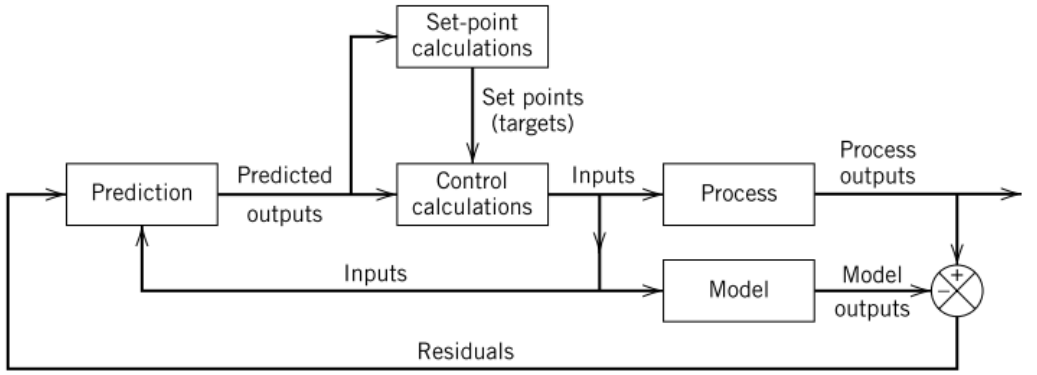
Predictive Models as Part of the Controller Architecture

- Model predictive controllers incorporate a dynamic process model as part of the architecture of the control algorithm
- The function of the dynamic model is to predict the future value of the measured process variable based on the current state of the process and knowledge of recent control actions. A control action is “recent” if the dynamic response it caused in the process is still in progress.
- If the predicted measured process variable does not match a desired set point, this future error enables corrective control actions to be taken immediately, and before the predicted problem actually occurs.
- Thus, the model predictive controller exploits process knowledge contained in the dynamic model to compute current control actions based on a predicted future.
- In theory, a perfect model can eliminate the negative influence of dead time on controller
- performance. In the practical world, MPC can certainly provide a performance benefit in the presence of large dead time. This benefit comes at a price, however. The designer must identify an appropriate dynamic model form, fit the model parameters to process data, and program the result into the control computer.

Block diagram representing the basic structure of MPC



MODEL PREDICTIVE CONTROL (MPC)

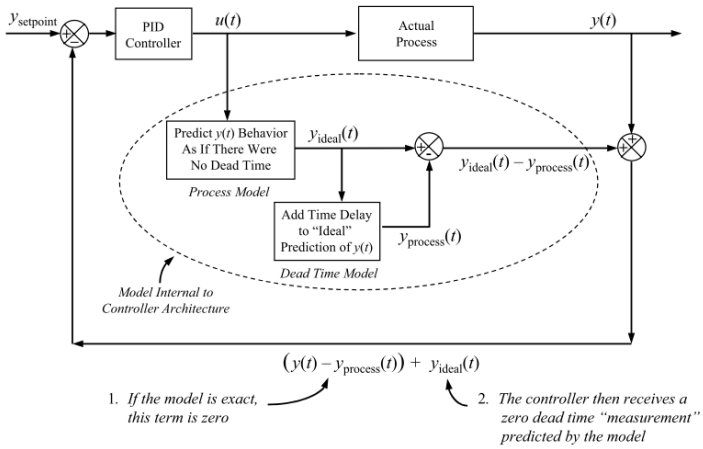


Model predictive control (MPC)

- is an effective means of dealing with large multivariable constrained control problems.
- The main idea is to choose the control action by repeatedly solving online an optimal control problem, aiming to minimize a performance criterion over a future horizon, possibly subject to constraints on the manipulated inputs and outputs.
- Future behavior is computed according to a model of the plant.
- Issues arise in guaranteeing closed-loop stability, handling model uncertainty, and reducing online computations.
- PID type controllers do not perform well when applied to systems with significant time delays.
- Model predictive control overcomes the debilitating problems of delayed feedback by using predicted future states of the output for control.
- Currently, some commercial controllers have Smith predictors as programmable blocks, but there are many other strategies with dead-time compensation properties.
- If there is no time delay, these algorithms usually collapse to the PID form.

The Smith predictor model based controller architecture

The Smith predictor architecture is comprised of an ideal (or no dead time) process model block and a separate dead time model block.



Internal models and robust controls

- Internal model control systems are characterized by a control device consisting of the controller and of a simulation of the process, the internal model.
- The internal model loop computes the difference between the outputs of the process and of the internal model
- This difference represents the effect of disturbances and of a mismatch of the model.

Internal model control characteristics

- If the process and the controller are (input-output) stable, and if the internal model is perfect, then the control system is stable.
- If the process and the controller are stable, if the internal model is perfect, if the controller is the inverse of the internal model, and if there is no disturbance, then perfect control is achieved.
- If the controller steady-state gain is equal to the inverse of the internal model steady-state gain, and if the control system is stable with this controller, then offset-free control is obtained for constant set points and output disturbances.

Summary

- Model-based controllers have demonstrated economic advantages over classical PID approaches, but they also have disadvantages.
- In order to develop a front-end model one must initiate substantial process disruptions (step tests) or initiate expensive engineering efforts.
- Often, additional or improved sensors are also required. Although SISO model-based controllers can be implemented in existing DCS or PLC microprocessors, often an additional microcomputer is also required for model-based controllers.
- The technology of the modeling, controller, tuning, and optimization approaches requires operator and process engineer training.
- Accordingly, MPC methods are recommended for difficult-to-control, multivariable, constrained, and/or economically critical processes.

Intelligent control systems

FUZZY CONTROL

What is fuzzy logic?

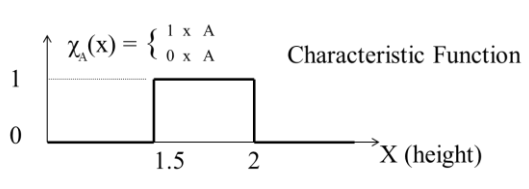
- Fuzzy logic is a superset of conventional (Boolean) logic that has been extended to handle the concept of partial truth -- truth values between "completely true" and "completely false".
- A form of knowledge representation suitable for notions that cannot be defined precisely, but which depend upon their context. It enables computerized devices to reason more like humans.
- Motivations:
- Alleviate difficulties in developing and analyzing complex systems encountered by conventional mathematical tools.
- Observing that human reasoning can utilize concepts and knowledge that do not have well-defined, sharp boundaries.

Basic Concepts of Fuzzy Logic

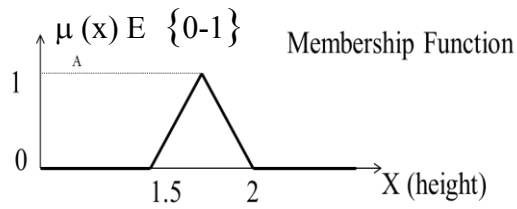
- Apparatus of fuzzy logic is built on:
  - Fuzzy sets: describe the value of variables
  - Linguistic variables: qualitatively and quantitatively described by fuzzy sets
  - Membership functions: constraints on the value of a linguistic variable
  - Fuzzy if-then rules: a knowledge

Fuzzy Set: Contain objects that satisfy imprecise properties of membership, a fuzzy set is a set with a smooth boundary.

Classical Set (Crisp) Contain objects that satisfy precise properties of membership:
Example: Set of heights from 1.5 to 2 meter



Fuzzy set example:
The set of heights in the region around 1.75 m



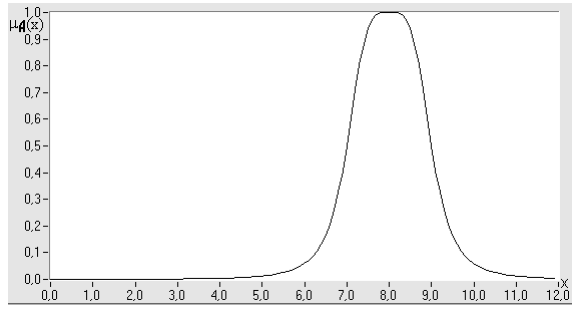
A fuzzy set is defined by a functions that maps objects in a domain of concern into their membership value in a set. Such a function is called the membership function.

Membership function

$\forall x \in \boldsymbol{X} : \mu(x) \geq O$,

- $\mu(x)$ the bigger the better x meets the evaluation criteria by an experienced professional.

•Example: Continous membership function

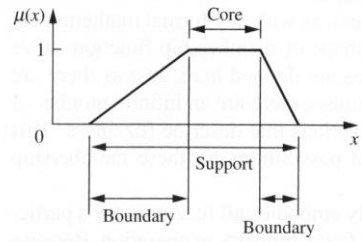


„approx. 8"

$$A = \left\{ \left(x, \mu_{\boldsymbol{A}}(x)\right) \middle| \mu_{\boldsymbol{A}}(x) = \left[1 + (x-8)^4\right]^{-1}; x \in \mathbf{X} \right\}$$

Features of the Membership Function

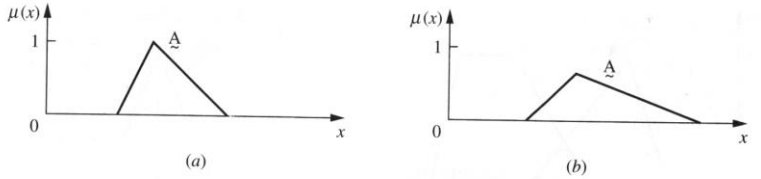
- Core: comprises those elements *x* of th universe such that $\mu_a(x) = 1$.
- Support : region of the universe that is characterized by nonzero membership.
- Boundary : boundaries comprise those elements *x* of the universe such that $0 < \mu_a(x) < 1$



$$\mathbf{S}(\mathbf{A}) = \left\{ x \in \mathbf{X} \middle| \mu_{\mathbf{A}}(x) > 0 \right\}, \mathbf{S}(\mathbf{A}) \subseteq \mathbf{X}$$

Features of the Membership Function (Cont.)

- Normal Fuzzy Set : at least one element x in the universe whose membership value is unity



Fuzzy sets that are normal (*a*) and subnormal (*b*).

- Height of a Fuzzy Set

$$hgt(A) = max[\mu_A(x)] \; \forall x \in X$$

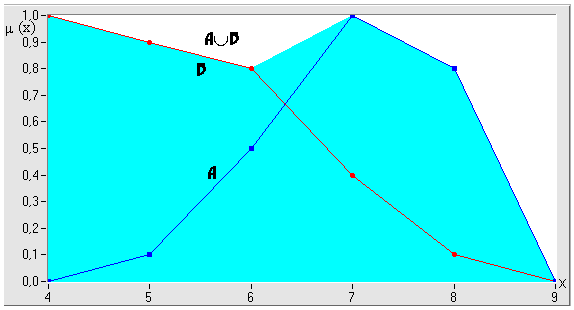
- Normalization: divide by height

Operations on Fuzzy Sets

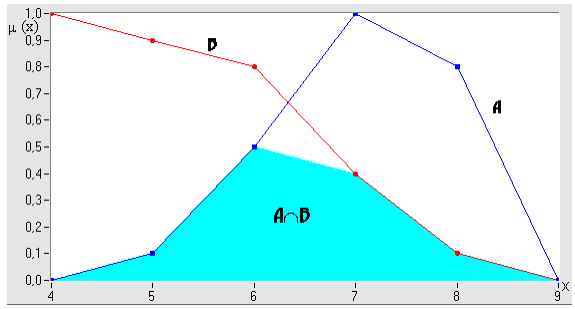
- Logical connectives:
  - Union
    - A U B : $\mu_{A \cup B}(x) = max[\mu_A(x), \mu_B(x)]$
  - Intersection
    - A . B : $\mu_{A \cap B}(x) = min[\mu_A(x), \mu_B(x)]$
  - Complementary
    - A ---> A^c : $\mu_A^c(x) = 1 - \mu_A(x)$

Operations on Fuzzy Sets

Union



Intersection



$$A_i = \{(x_i, \mu_{Ai}(x),\ x \in X\} \quad A_i \in P(X), \quad i=1 \ldots n.$$

- **Fuzzy-AND**

$$\mu_{and} = \delta \cdot \min\left(\mu_{A_i}(X)\right) + \frac{1-\delta}{n} \cdot \sum \mu_{A_i}(X)$$

i=1 … n.

- **Fuzzy-OR**

$$\mu_{or} = \delta \cdot \max\left(\mu_{A_i}(X)\right) + \frac{1-\delta}{n} \cdot \sum \mu_{A_i}(X)$$

i=1 … n.

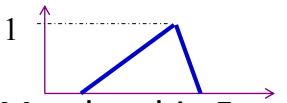
- **γ-union**

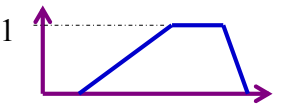
$$\mu_{\gamma}(x) = \left[\prod_i \mu_{A_i}(x)\right]^{1-\gamma} \cdot \left[1 - \prod_i \left(1-\mu_{Ai}(x)\right)\right]^{\gamma}$$

i=1 … n.

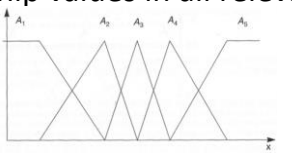
Guidelines for membership function design

- Always use parameterizable membership functions. Do not define a membership function point by point.
  - Triangular and Trapezoid membership functions are sufficient for most practical applications!





- Designing Antecedent Membership Functions
  - Each Membership function overlaps only with the closest neighboring membership functions;
  - For any possible input data, its membership values in all relevant fuzzy sets should sum to 1 (or nearly)



Fuzzy control

- Fuzzy systems have been applied to a wide variety of fields ranging from control, signal processing, communications, integrated circuit manufacturing, and expert systems to business, medicine, psychology, etc.

- the most significant applications have concentrated on control problems

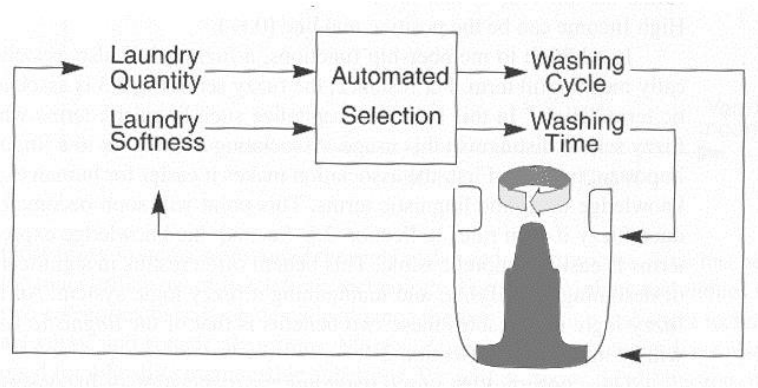
Fuzzy systems can be used either as

- open-loop controllers
  - the fuzzy system usually sets up some control parameters and then the system operates according to these control parameters.
  - Many applications of fuzzy systems in consumer electronics.
- closed-loop controllers
  - the fuzzy system measures the outputs of the process and takes control actions on the process continuously.
  - Applications of fuzzy systems in industrial processes.

Fuzzy control examples

- Washing machine
  - (https://www.samsung.com/in/support/home-appliances/what-is-fuzzy-logic-in-a-washing-machine/)
- Most fuzzy logic machines feature 'onetouch control.' Equipped with energy saving features, these consume less power and are worth paying extra for if you wash full loads more then three times a week. Inbuilt sensors monitor the washing process and make corrections to produce the best washing results.
- fuzzy logic controls the washing process, water intake, water temperature, wash time, rinse performance, and spin speed. More sophisticated machines weigh the load (so you can't overload the washing machine), advise on the required amount of detergent, assess cloth material type and water hardness, and check whether the detergent is in powder or liquid form. Some machines even learn from **past experience**, **memorizing programs** and adjusting them to **minimize running costs**.
- The **fuzzy logic** checks for the extent of dirt and grease, the amount of soap and water to add, direction of spin, and so on. The machine rebalances washing load to ensure correct spinning. Else, it reduces spinning speed if an imbalance is detected. Even distribution of washing load reduces spinning noise. **Neuro fuzzy logic** incorporates optical sensors to sense the dirt in water and a **fabric sensor** to detect the type of fabric and accordingly adjust wash cycle.

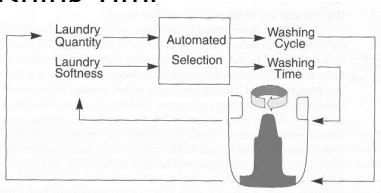
Simplified fuzzy control modell

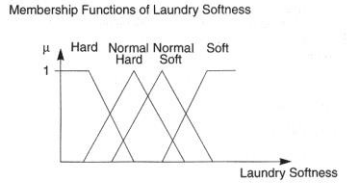


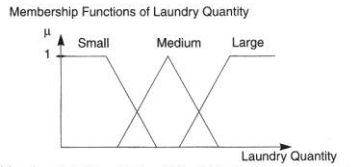
* Fuzzy Logic: Intelligence, control, and Information, J. Yen and R. Langari, Prentice Hall

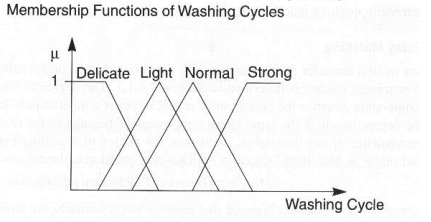
Fuzzy control examples

- Inputs
  - —Laundry Softness
  - —Laundry Quantity
- Outputs
  - —Washing Cycle
  - —Washing Time









Fuzzy rule relates fuzzy sets

IF *X is A*, THEN *Y is B*

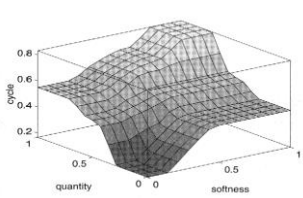
- A and B are fuzzy sets and subset of X and Y variables

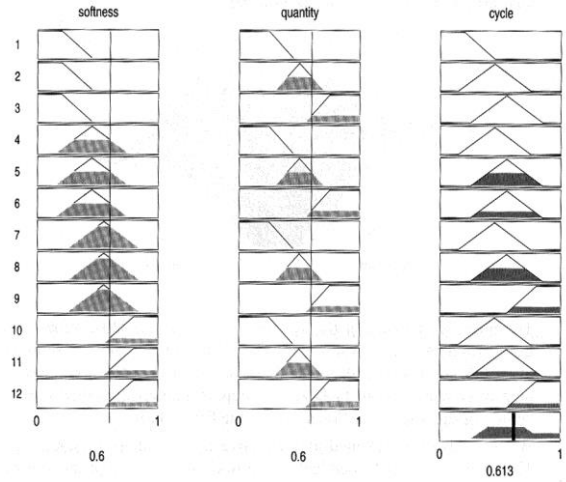
- Fuzzy logic is a superset of conventional (Boolean) logic that has been extended to handle the concept of partial truth -- truth values between "completely true" and "completely false".

Rule base

Quantity / Softness	Small	Medium	Large
Soft	Delicate	Light	Normal
Normal Soft	Light	Normal	Normal
Normal Hard	Light	Normal	Strong
Hard	Light	Normal	Strong

Control Surface View, Rule View
AND: min, β_i:min aggregation:max





Load Sway Example

Cranes load and unload containers to/from ships

- Load *always* sways
- Swaying load may hit other containers
- Swaying load cannot be released

Load Sway Example

Ineffective solutions:

1. Position the crane over the target and wait for sway to subside
2. Move container slow, so no sway occurs
3. Use additional links (cables) to fix the load in place and prevent sway

Most cranes are still controlled by human operators that have skills to quickly compensate the swaying of the load!

Why the problem was not solved by the classical control theory?

PID control:

- The problem is non-linear (for example, sway minimization is important only close to the target)
- Many unknown variables

Model-based control:

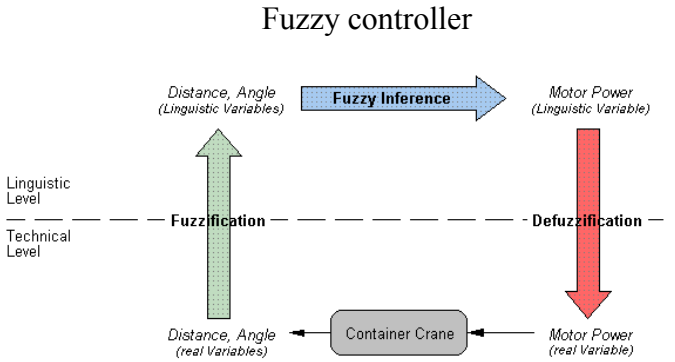
- Fifth-degree differential equation that describes the system
- Works in theory, does not work in practice (too many simplifications and unknown variables)

Load Sway Example

Heuristic strategy:

- Start with medium power.
- If you get started and you are still far away from target, adjust the motor power so that the container gets a little behind the crane head.
- If you are closer to the target, reduce speed so the container gets a little ahead of the crane head.
- When the container is very close to the target position, power up the motor.
- When the container is over the target and the sway is zero, stop the motor.

Load Sway Example

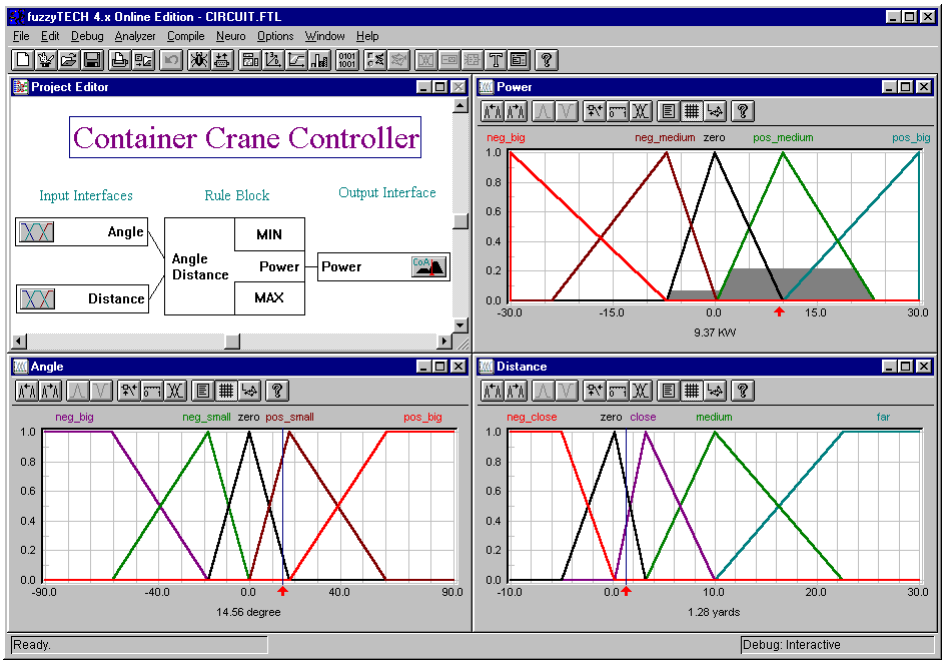


IF…THEN… Rules

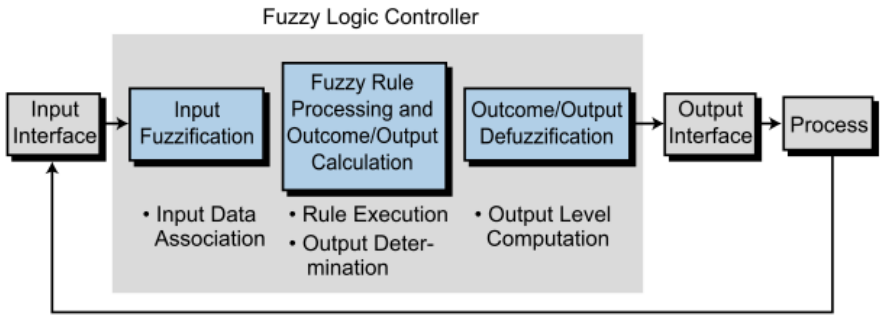
- 1. IF Distance = far AND Angle = zero THEN Power = pos_medium
- 2. IF Distance = far AND Angle = neg_small THEN Power = pos_big
- 3. IF Distance = far AND Angle = neg_big THEN Power = pos_medium
- 4. IF Distance = medium AND Angle = neg_small THEN Power = neg_medium
- 5. IF Distance = close AND Angle = pos_small THEN Power = pos_medium
- 6. IF Distance = zero AND Angle = zero THEN Power = zero

Spreadsheet Rule Editor - RB1

	IF		THEN	
	Angle	Distance	DoS	Power
1	zero	far	1.00	pos_medium
2	neg_small	far	1.00	pos_big
3	neg_big	far	1.00	pos_medium
4	neg_small	medium	1.00	neg_medium
5	pos_small	close	1.00	pos_medium
6	zero	zero	1.00	zero

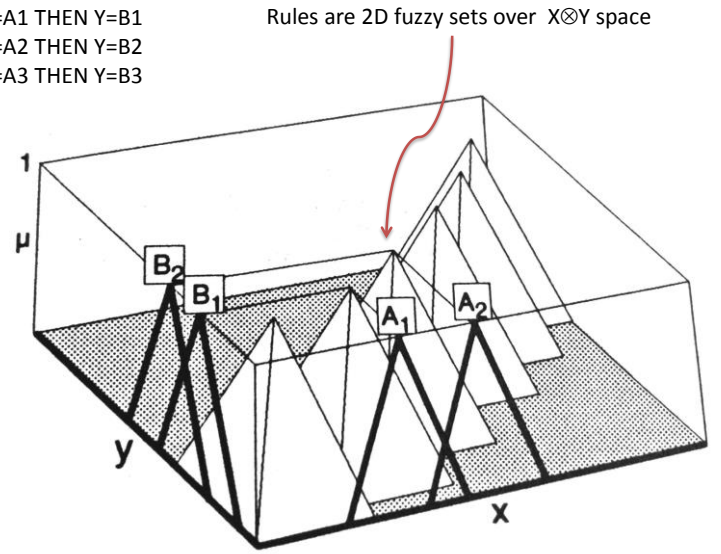


Fuzzy logic controller operation



Rule's representation

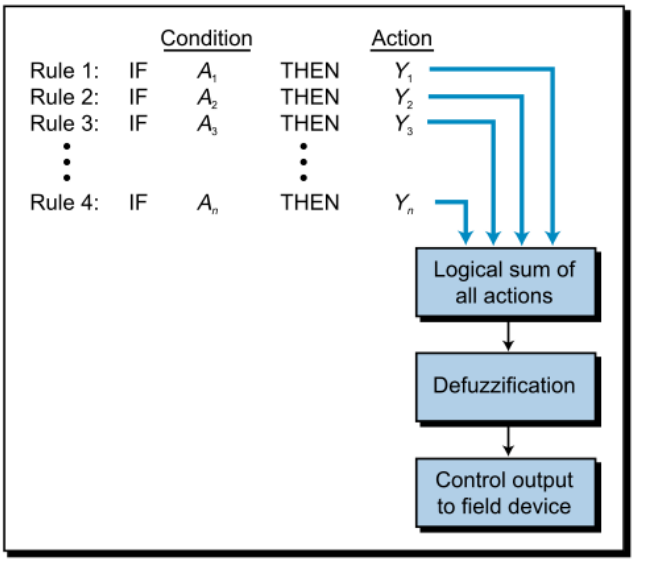
IF X=A1 THEN Y=B1
IF X=A2 THEN Y=B2
IF X=A3 THEN Y=B3
…



Rule Evaluation

- Fuzzy logic is based on the concept that most complicated problems are formed by a collection of simple problems and can, therefore, be easily solved.
- Fuzzy logic uses a reasoning, or inferencing, process
- composed of IF...THEN rules, each providing a response or outcome.
- Basically, a rule is activated, or triggered, if an input condition satisfies the IF part of the rule statement.
- This results in a control output based on the THEN part of the rule statement.
- In a fuzzy logic system, many rules may exist, corresponding to one or more IF conditions

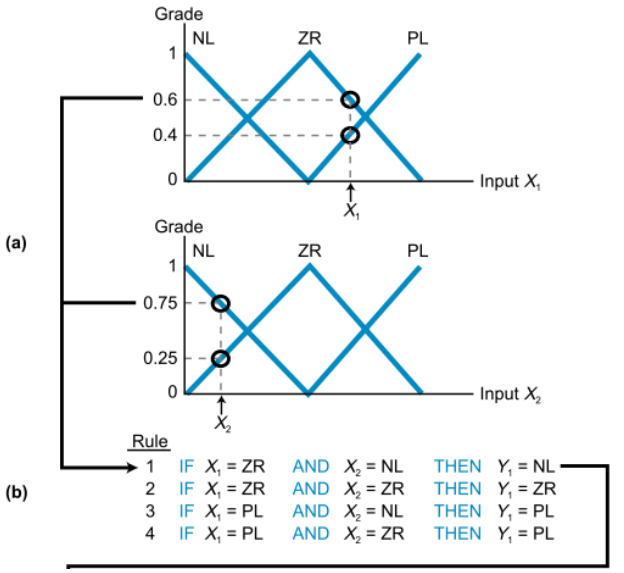
Output from multiply rules



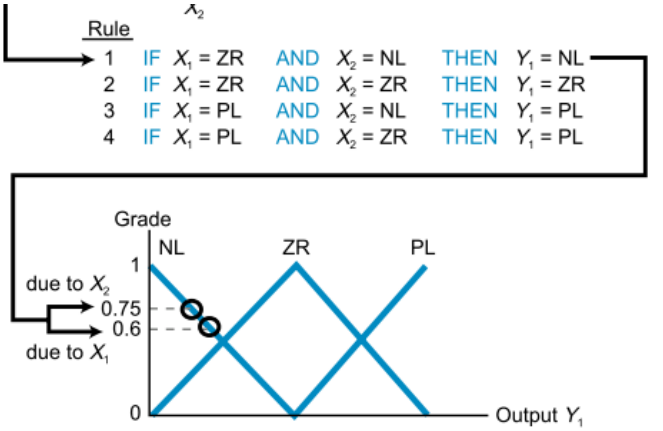
A rule may also have several input conditions, which are logically linked in either an AND or an OR relationship to trigger the rule's outcome

		Condition		Action
Rule 1:	IF	A_1 AND B_1 AND C_1	THEN	Y_1
Rule 2:	IF	A_2 OR B_2	THEN	Y_2
Rule 3:	IF	(A_3 AND B_3) OR C_3	THEN	Y_3

An example of two fuzzy inputs, X_1 and X_2 , and one fuzzy output, Y_1 .

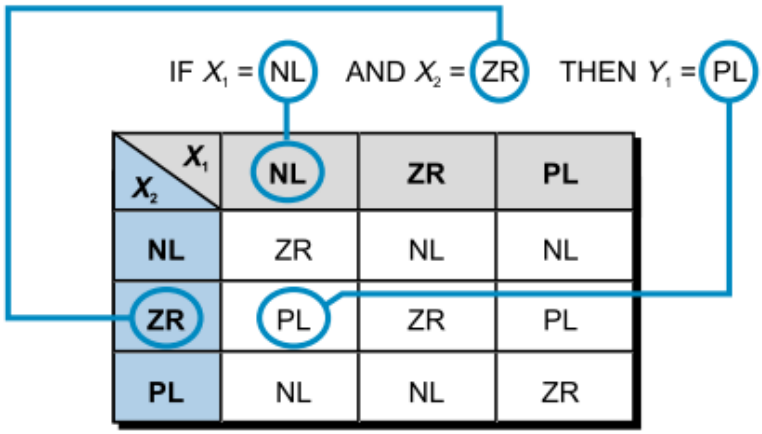


Fuzzy processing example



The logical AND function specifies that the fuzzy controller will select the lowest grade (0.6NL) as the outcome of the rule.

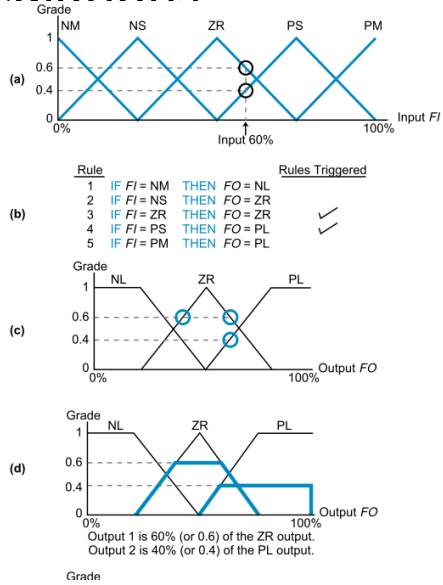
Fuzzy logic rules with two inputs are often represented in matrix form to represent AND conditions.



X_2 \ X_1	NL	ZR	PL
NL	ZR	NL	NL
ZR	PL	ZR	PL
PL	NL	NL	ZR

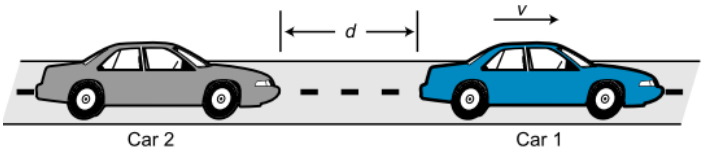
Fuzzy Outcome Calculations

- Once a rule is triggered, meaning that the input data belongs to a membership function that satisfies the rule's IF statement, the rule will generate an output outcome.
- This fuzzy output is composed of one or more membership functions (with labels), which have grades associated with them.
- The outcome's membership function grade is affected by the grade level of the input data in its input membership function.



Example

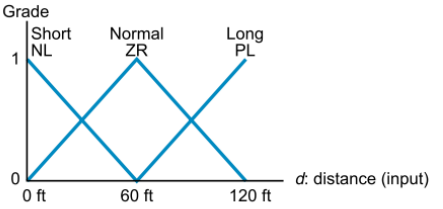
- Two cars separated by a distance *d* , which can range between 0 and 120 feet.
- Car 1 travels at a speed of v, ranging from 0 to 80 mph.
- Depending on the speed and distance, car 2 has several braking options (B) ranging from light to hard if car 1 slows down or stops.

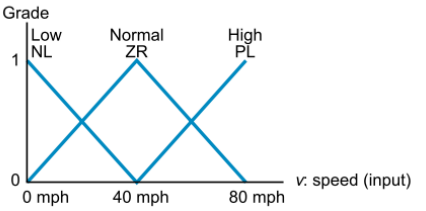


d = distance between cars
v = velocity (speed) of car 1
B = braking strength (function of *d* and *v*)

Two input fuzzy sets: distance and speed

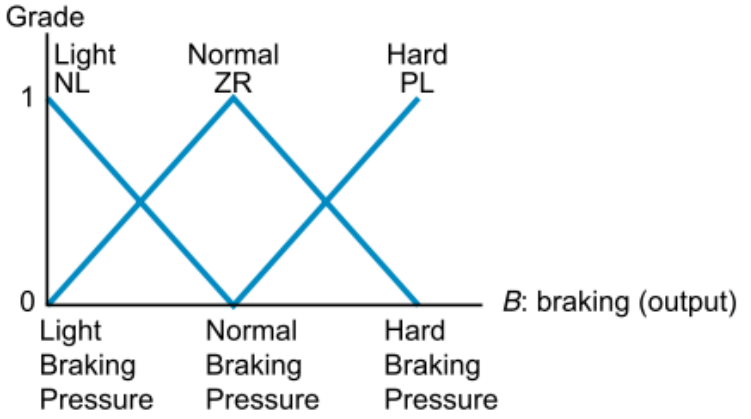
- The distance fuzzy set ranges from 0 to 120 feet
- The speed fuzzy set ranges from 0 to 80 mph.





The output fuzzy sets (braking strength)

ranges from light braking to hard braking



The fuzzy system's rules

- Example rules:
- IF the distance between the two cars is long and the speed is normal, THEN brake lightly.
- IF d = PL AND v = ZR THEN B = NL

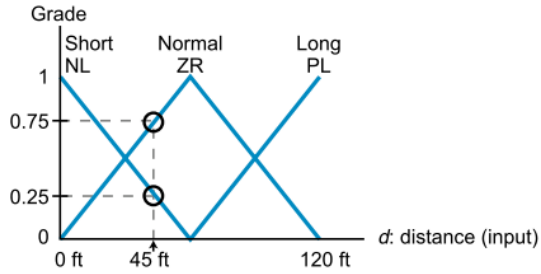
	Fuzzy Rules
1	IF *d* = NL AND *v* = NL THEN *B* = ZR
2	IF *d* = NL AND *v* = ZR THEN *B* = PL
3	IF *d* = NL AND *v* = PL THEN *B* = PL
4	IF *d* = ZR AND *v* = NL THEN *B* = NL
5	IF *d* = ZR AND *v* = ZR THEN *B* = ZR
6	IF *d* = ZR AND *v* = PL THEN *B* = PL
7	IF *d* = PL AND *v* = NL THEN *B* = NL
8	IF *d* = PL AND *v* = ZR THEN *B* = NL
9	IF *d* = PL AND *v* = PL THEN *B* = ZR

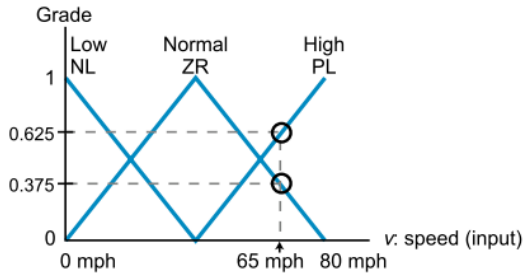
Rules in matrix form

Distance *d* / Speed *v*	Short NL	Normal ZR	Long PL
Low NL	ZR (Brake Normal)	NL (Brake Light)	NL (Brake Light)
Normal ZR	PL (Brake Hard)	ZR (Brake Normal)	NL (Brake Light)
High PL	PL (Brake Hard)	PL (Brake Hard)	ZR (Brake Normal)

Let: inputs d =45 ft and v = 65 mph.

- Each input triggers (crosses) two membership functions:
- input d crosses membership functions ZR and NL;
- input v crosses membership functions PL and ZR.





These inputs trigger four rules: active rules (firing rules)

0.25 0.75

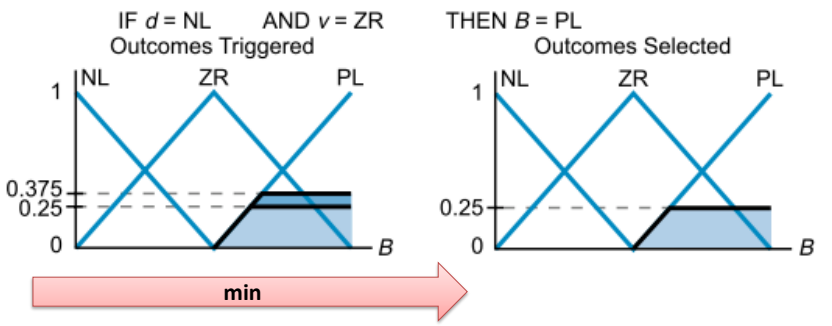
Speed v \ Distance d	Short NL	Normal ZR	Long PL
Low NL	ZR (Brake Normal)	NL (Brake Light)	NL (Brake Light)
Normal ZR	PL (Brake Hard)	ZR (Brake Normal)	NL (Brake Light)
High PL	PL (Brake Hard)	PL (Brake Hard)	ZR (Brake Normal)

0.375

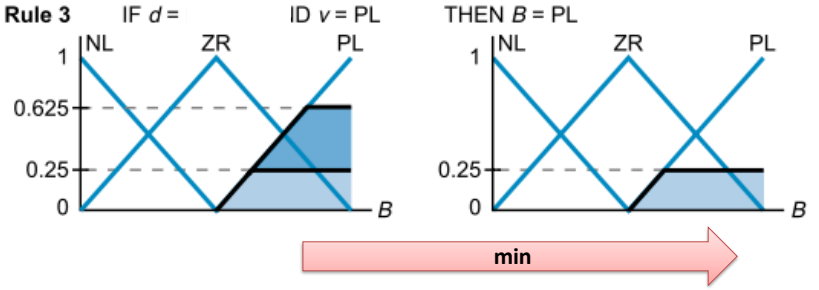
0.625

Active rules

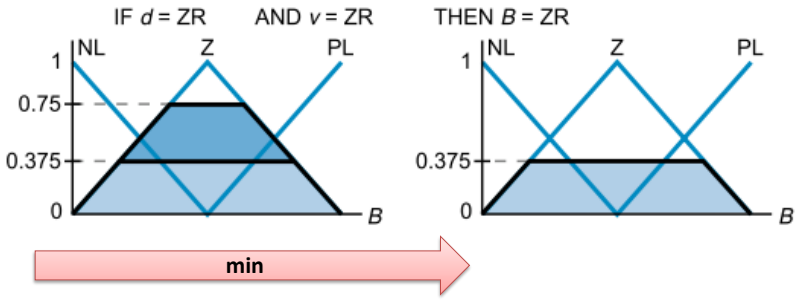
The rules' outputs will correspond to the smallest input grade value (AND)



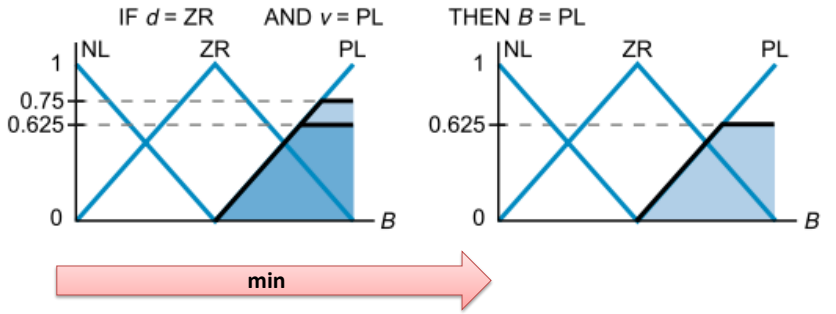
Outcome of the first active rule



Outcome of the second active rule

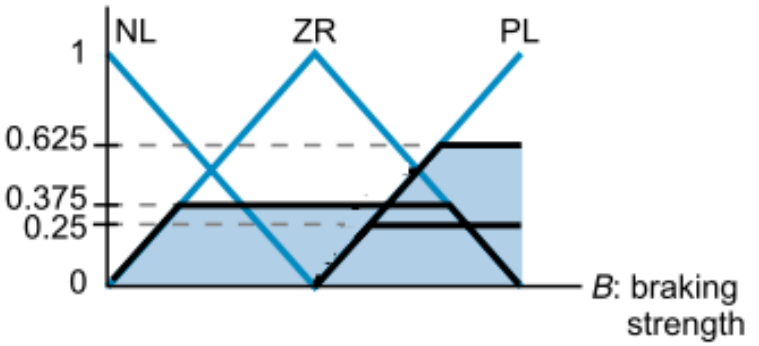


Outcome of the third active rule



Outcome of the fourth active rule

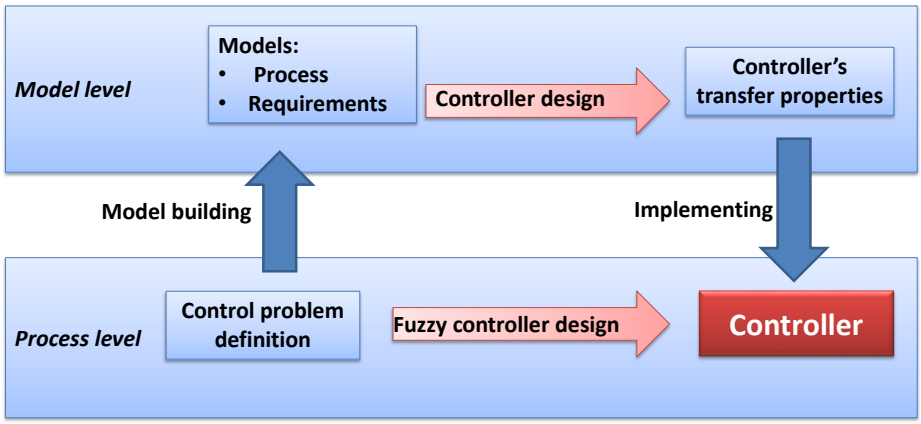
Fuzzy output: union of active rule's outputs



Expert knowledge

- Fuzzy logic requires knowledge in order to reason. This knowledge, which is provided by a person who knows the process or machine (the expert), is stored in the fuzzy system.
- For example, if the temperature rises in a temperature- regulated batch system, the expert may say that the steam valve needs to be turned clockwise a "little bit."
- A fuzzy system may interpret this expression as a 10-degree clockwise rotation that closes the current valve opening by 5%.
- As the name implies, a description such as a "little bit" is a fuzzy description, meaning that it does not have a definite value.

Controller design procedures:
Classic controller vs. fuzzy controller



Intelligent control systems

FROM DIGITAL PI CONTROL TO FUZZY PI CONTROL

PID digital

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

t=nT

$$u_n = K_P\left(e_n + \frac{1}{T_I}\sum_{i=1}^{n} e_i T + T_D\frac{e_n - e_{(n-1)}}{T}\right)$$

substraction

$$u_{(n-1)} = K_P\left(e_{rn-1)} + \frac{1}{T_I}\sum_{i=1}^{n-1} e_i T + T_D\frac{e_{(n-1)} - e_{(n-2)}}{T}\right)$$

$$\Delta u_n = u_n - u_{(n-1)} = K_P\left(\left(e_n - e_{(n-1)}\right) + \frac{T}{T_I}e_n + \frac{T_D}{T}\left(e_n - 2e_{(n-1)} + e_{(n-2)}\right)\right)$$

$$u_n = u_{(n-1)} + K_P\left(\left(e_n - e_{(n-1)}\right) + \frac{T}{T_I}e_n + \frac{T_D}{T}\left(e_n - 2e_{(n-1)} + e_{(n-2)}\right)\right)$$

Digital PI-algorithm

$$u_n = u_{(n-1)} + K_P\left(\left(e_n - e_{(n-1)}\right) + \frac{T}{T_I}e_n\right)$$

$$u_n = u_{(n-1)} + \Delta u_n$$

$$\Delta u_n = K_P\left(\left(e_n - e_{(n-1)}\right) + \frac{T}{T_I}e_n\right)$$

The controller output change is proportional to

Error change: $\Delta e_n = e_n - e_{n-1}$

Error: e_n

PI control strategy

$$\Delta u_n = K_P\left(\left(e_n - e_{(n-1)}\right) + \frac{T}{T_I}e_n\right)$$

Error change: $\Delta e_n = e_n - e_{n-1}$

Error: e_n

- If error is pos then increase the controller output
- If error is negative then decrese the controlller output
- If error change is pos then increase the controller output
- If error change is neg then increase the controller output

Write in matrix form

$$\Delta u_n = K_P\left(\left(e_n - e_{(n-1)}\right) + \frac{T}{T_I}e_n\right)$$

If error is pos then increase the controller output!
If error is negative then decrese the controlller output!
If error change is pos then increase the controller output!
If error change is neg then increase the controller output!

Consider two variable: e_n and De_n:
IF e_n=pos AND De_n=Pos THEN DU_n = pos
IF e_n=neg AND De_n=neg THEN DU_n = neg
Etc.:

De_n / E_n	neg	zero	pos
neg	neg	neg	
zero	neg	zero	pos
pos		pos	poz

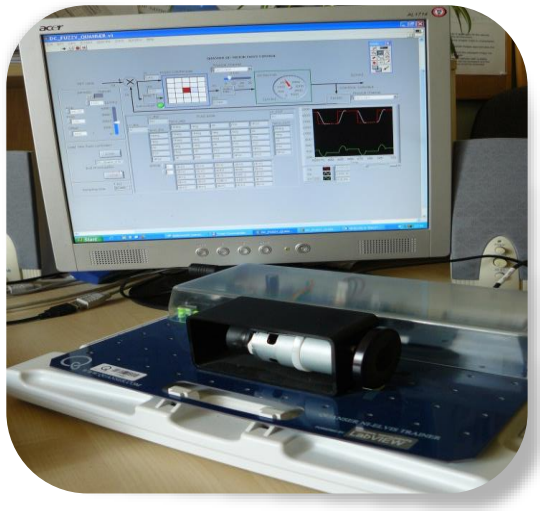
Write in matrix form

If error is pos then increase the controller output
If error is negative then decrese the controlller output
If error change is pos then increase the controller output
If error change is neg then increase the controller output

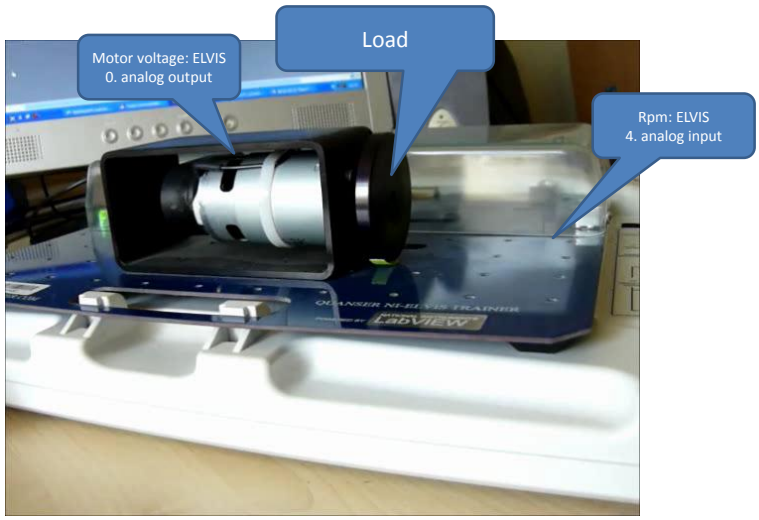
De_n E_n	neg	zero	pos
neg	neg	neg	Neg+pos=zero
zero	neg	zero	pos
pos	Pos+neg=zero	pos	poz

Rule base of a Fuzzy PI controller !

DC motor fuzzy speed control



DCmotor



System Schematic

A schematic of the DCMCT system interfaced with a DAQ device is provided in Figure B.2.3.

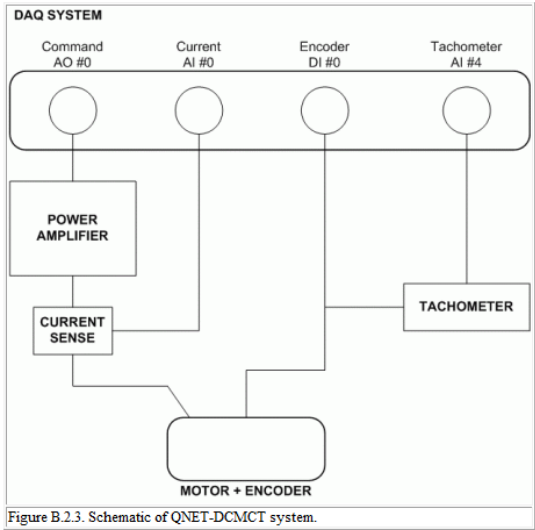
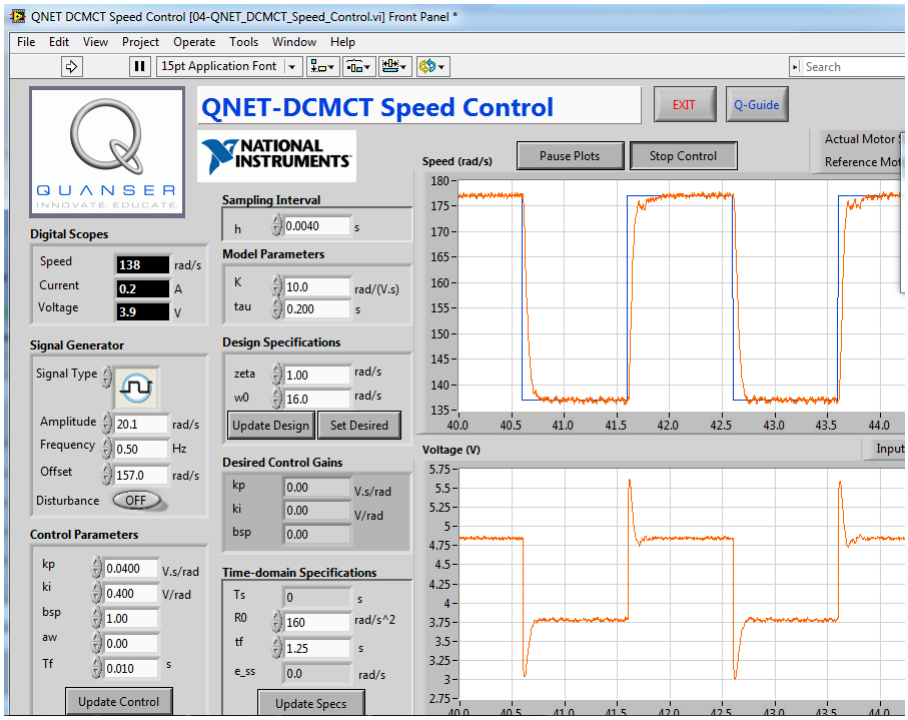


Figure B.2.3. Schematic of QNET-DCMCT system.

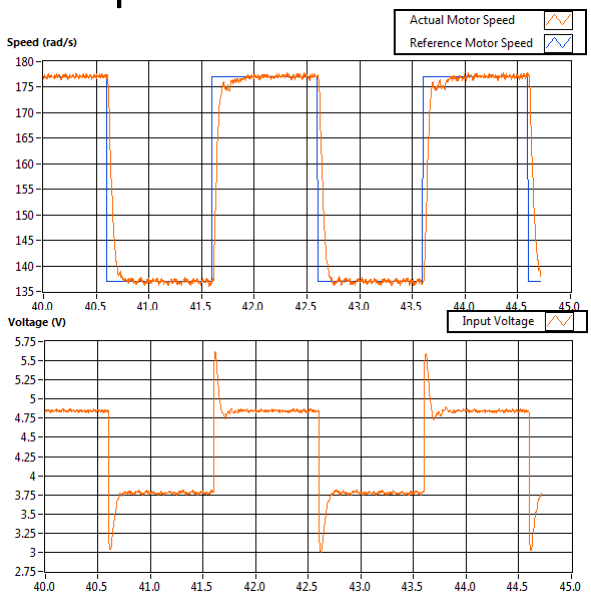
The specifications on the DCMCT system sensors are given in Table B.2.3.

Description	*Value*	*Unit*
Current Sense:		
Current calibration	1.0	A/V
Current sense resistor	0.1	ohms
Encoder:		
Encoder line count	1024	lines/rev
Encoder resolution (in quadrature mode)	0.0879	deg/count
Encoder type	TTL	
Encoder signals	A,B	
Tachometer:		
Tachometer calibration at QNET A/D input	1050	RPM/V

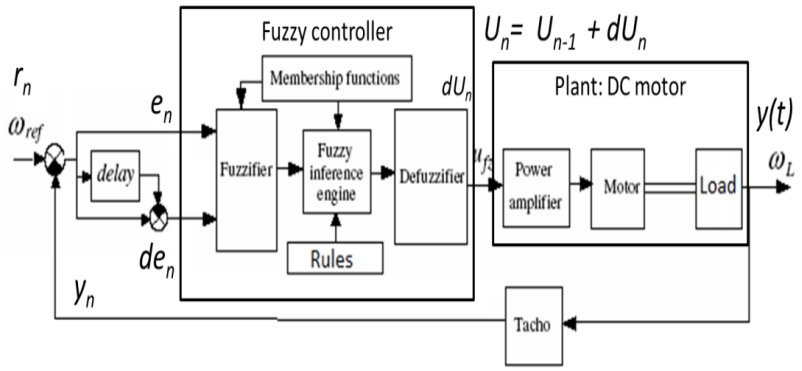
Table B.2.3. DCMCT sensor parameter specifications



Speed PI control

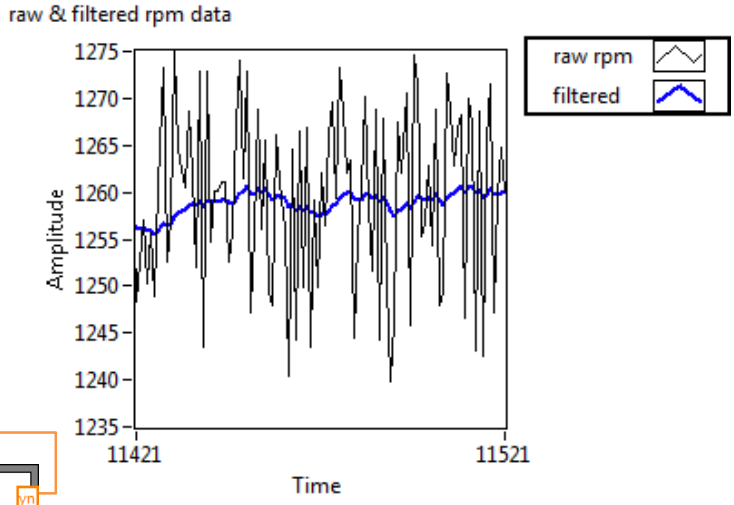


Block diagram of the fuzzy motor control

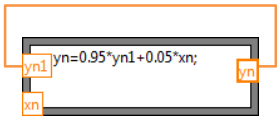


Measured rpm data must be filtered (smoothed) before the controller

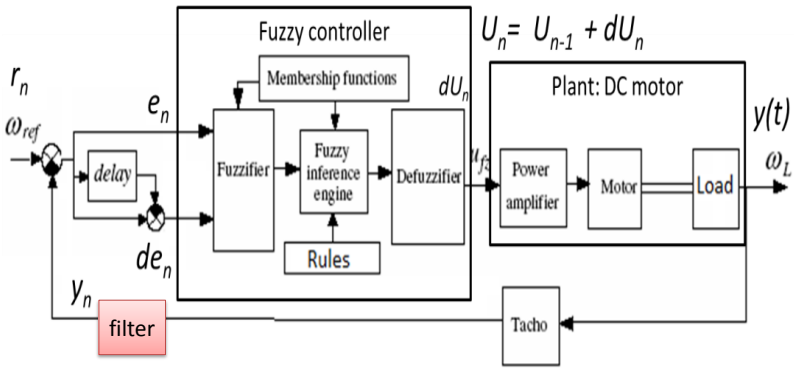
± 20 rpm scattering



Filter: weighted avareage

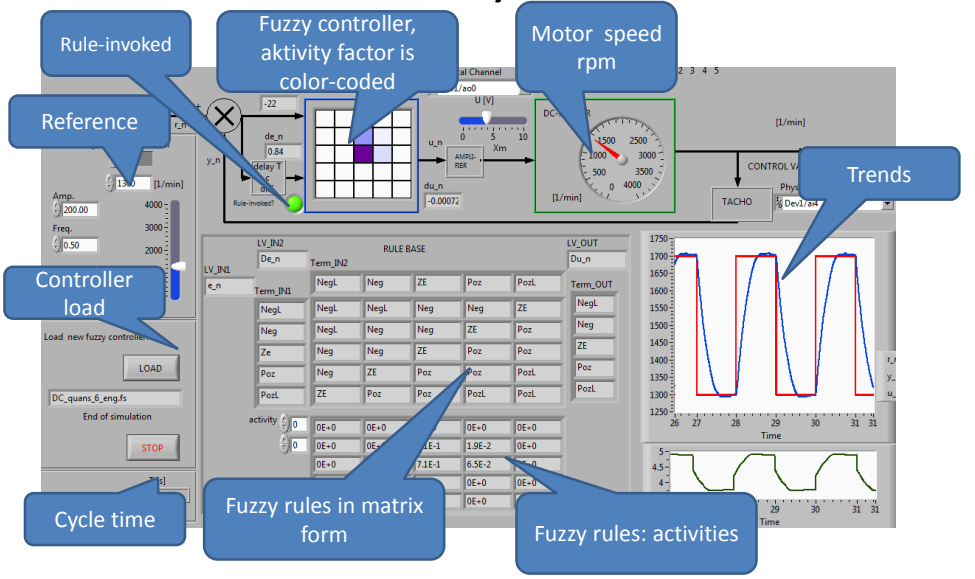


Block diabram

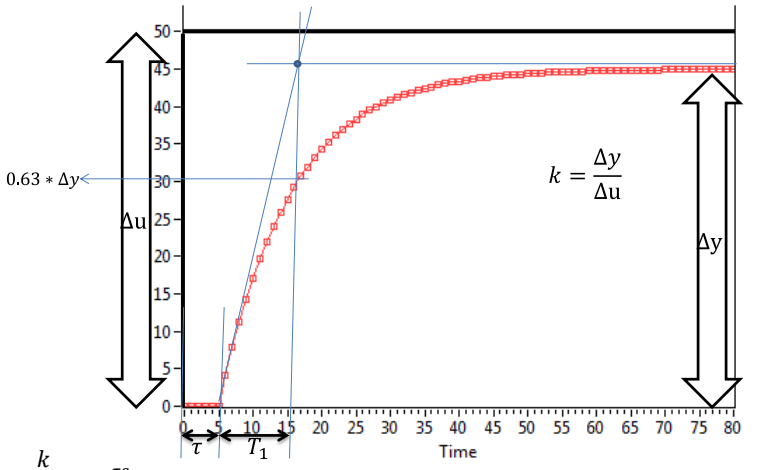


Filtering means a plus first order component in a control loop!

Motor Fuzzy PI-control

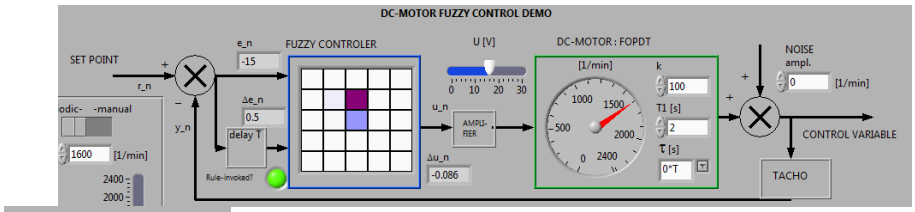


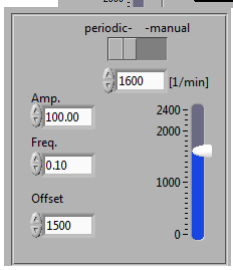
Parameter estimation from the step response of the system



$$P(s) = \frac{k}{T_1 s + 1} e^{-\tau s}$$

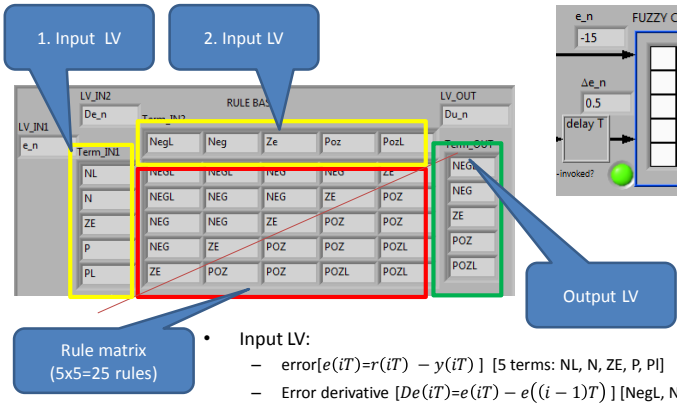
Block diagram in LV program





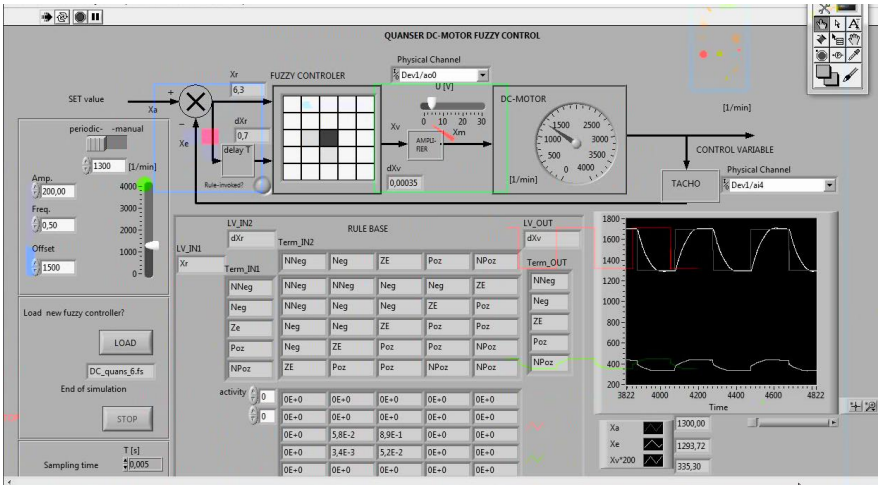
- Desired set point (reference) can be set manually or periodically (square)
- The amplitude of the disturbance can be set

Fuzzy PI-controller

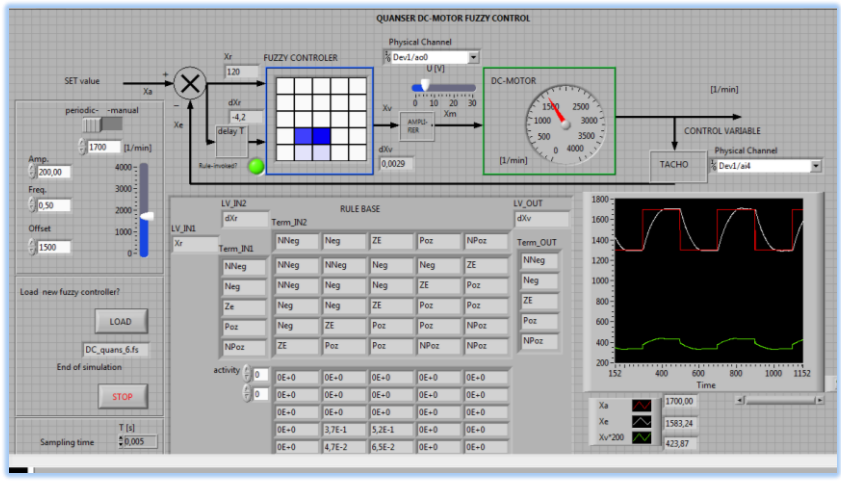


- Input LV:
  - error[$e(iT)$=$r(iT)$ $-$ $y(iT)$] [5 terms: NL, N, ZE, P, PI]
  - Error derivative [$De(iT)$=$e(iT) - e\big((i-1)T\big)$] [NegL, Neg, Ze, Poz, PozL]
- Output LV:
  - Change in motor voltage [$\Delta X_v(iT)$] [NEGL, NEG, ZE, POZ, POZL]
- Rulematrix: symmetrical diagonally
- Example: IF e_n = NL AND DE-n = NEG THEN Du_n = NegL

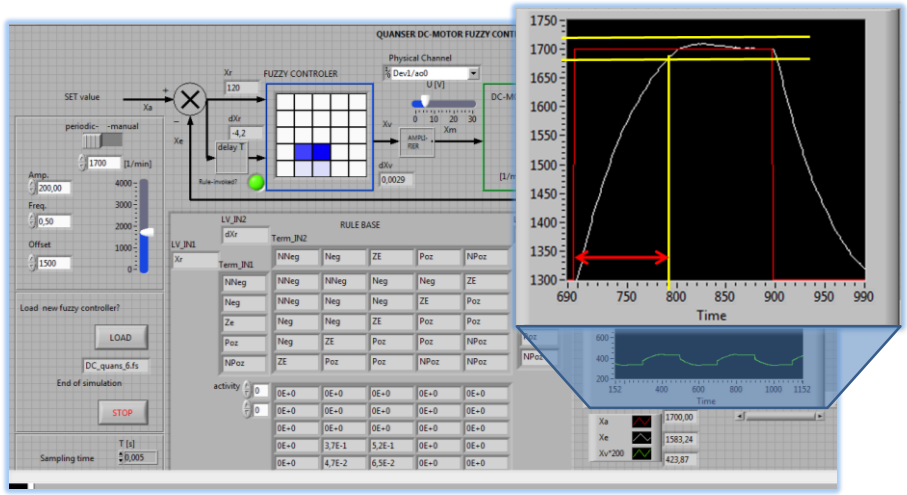
Example I.



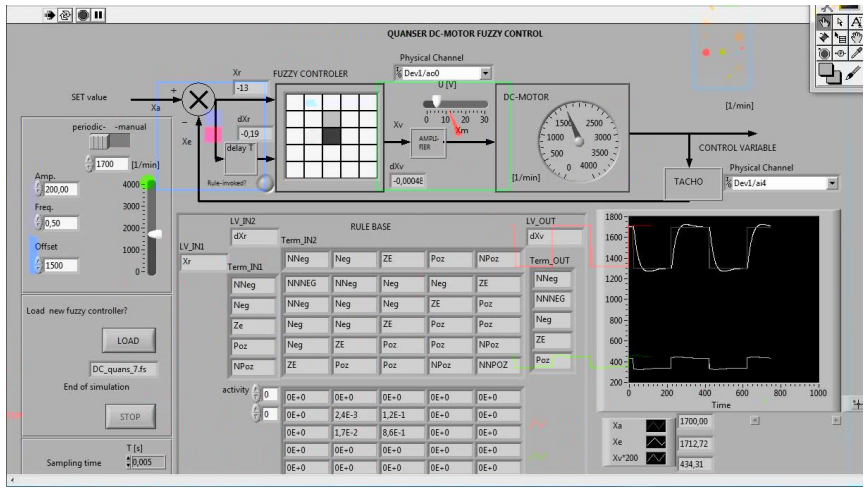
Settling time, example I.



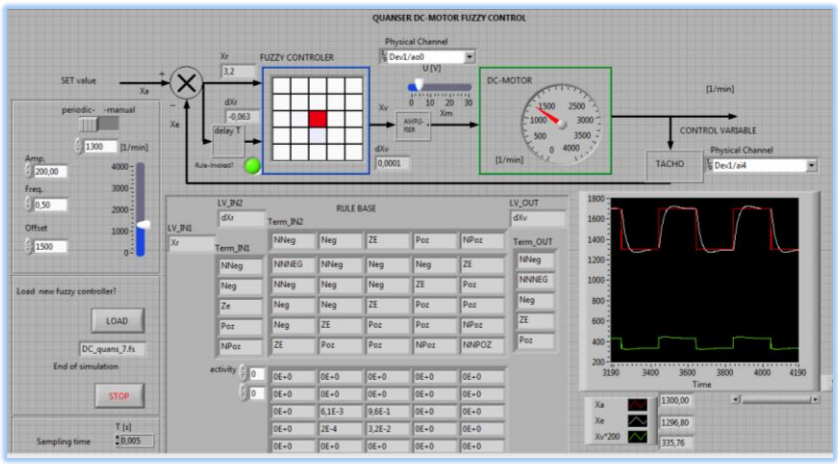
Settling time, example I.



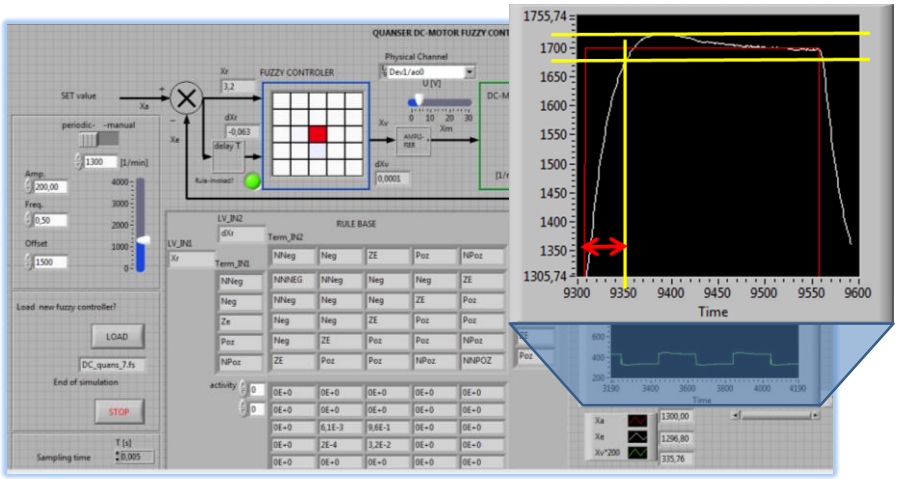
Example II. 7 terms of output LV



Settling time, example II.

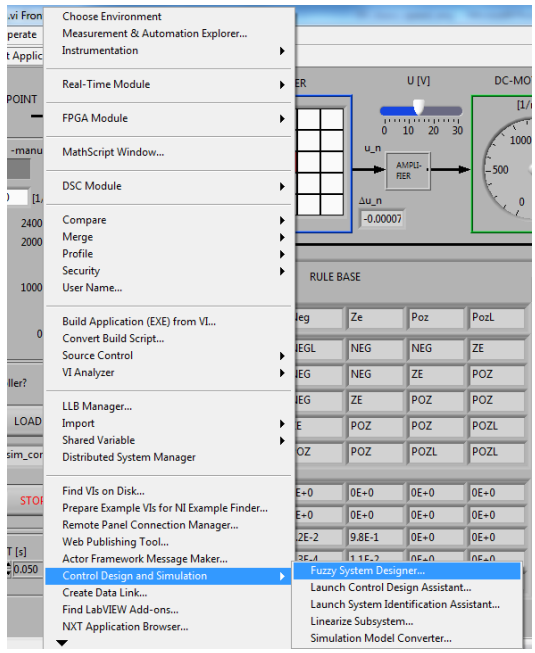


Settling time, example II.

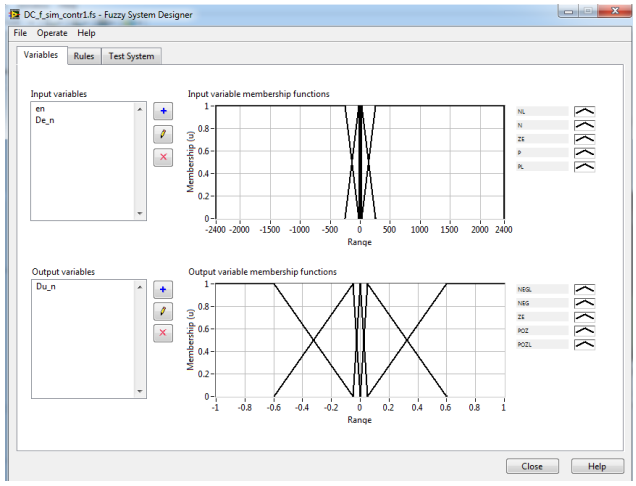


LabVIEW Fuzzy system designer in tools menu

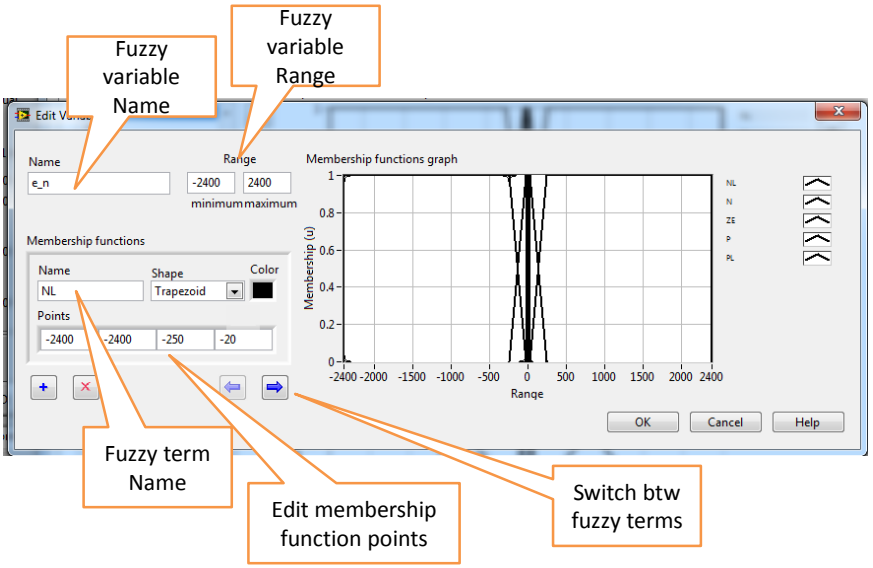
The fuzzy controller can be designed in a separated window (GUI).



Fuzzy system designer



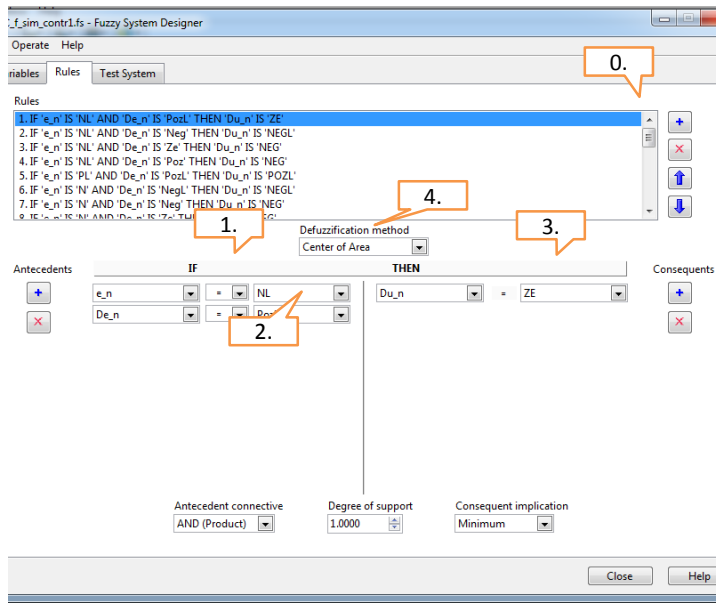
Edit variable window



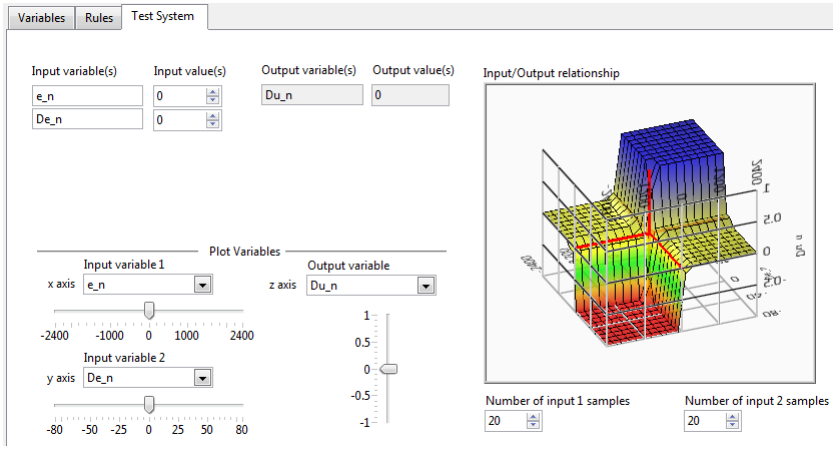
Rule editor

Rules edited in IF … THEN …. Form according to the next steps:

0. Press ➕
1. Select antecedent terms
2. Select consequent term
3. The rule defined in list
4. Select defuzzification method
5. Save

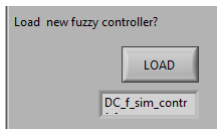


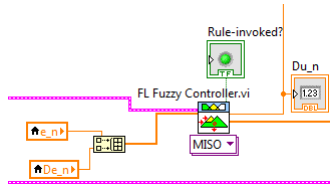
I/O surface of a fuzzy controller



LabVIEW Fuzzy system designer

- The fuzzy controller have to be saved in a file
- The LV program calls this file





Rules are only listed in Fuzzy system designer

In the simulation we can see rules in matrix form

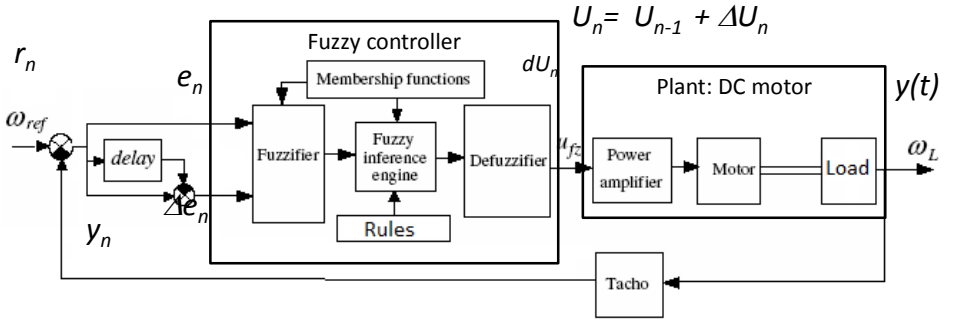
Rules

1. IF 'e_n' IS 'NL' AND 'De_n' IS 'PozL' THEN 'Du_n' IS 'ZE'
2. IF 'e_n' IS 'NL' AND 'De_n' IS 'Neg' THEN 'Du_n' IS 'NEGL'
3. IF 'e_n' IS 'NL' AND 'De_n' IS 'Ze' THEN 'Du_n' IS 'NEG'
4. IF 'e_n' IS 'NL' AND 'De_n' IS 'Poz' THEN 'Du_n' IS 'NEG'
5. IF 'e_n' IS 'PL' AND 'De_n' IS 'PozL' THEN 'Du_n' IS 'POZL'
6. IF 'e_n' IS 'N' AND 'De_n' IS 'NegL' THEN 'Du_n' IS 'NEGL'
7. IF 'e_n' IS 'N' AND 'De_n' IS 'Neg' THEN 'Du_n' IS 'NEG'

LV_IN1 e_n	LV_IN2 De_n	RULE BASE				LV_OUT Du_n
Term_IN1	Term_IN2	NegL	Neg	Ze	Poz	PozL
NL		NEGL	NEGL	NEG	NEG	ZE
N		NEGL	NEG	NEG	ZE	POZ
ZE		NEG	NEG	ZE	POZ	POZ
P		NEG	ZE	POZ	POZ	POZL
PL		ZE	POZ	POZ	POZL	POZL

Term_OUT: NEGL, NEG, ZE, POZ, POZL

Fuzzy PI control simulation: Block diagram

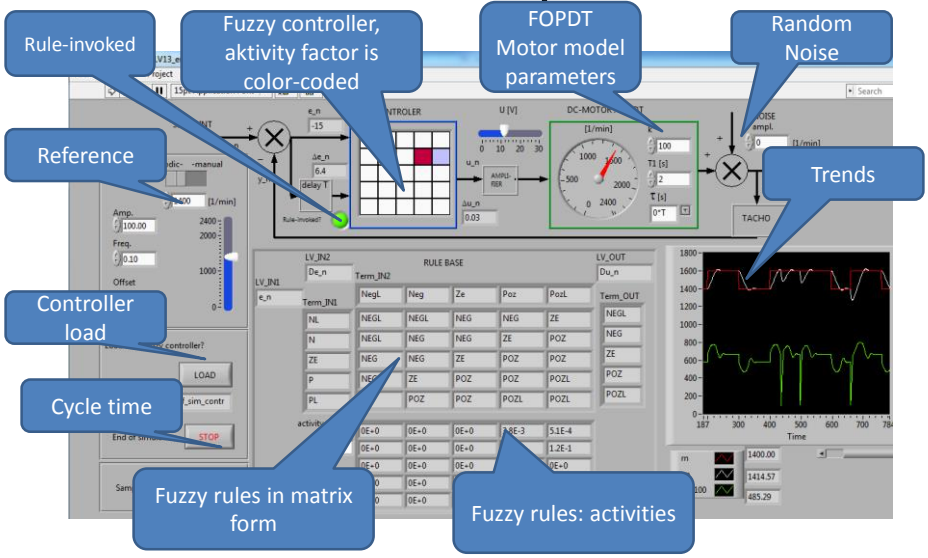


DC motor transfer characteristic: first-order + time delay: FOPTD system:

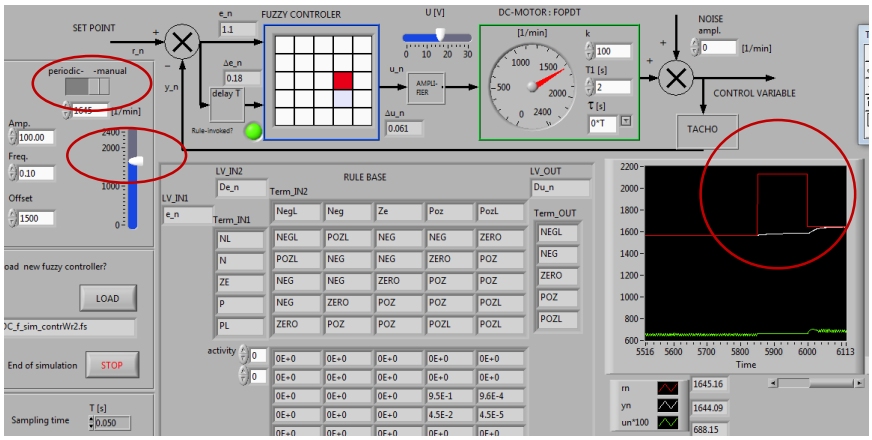
$$P(s) = \frac{k}{T_1 s + 1} e^{-\tau s}$$

The model parameters have to be estimated for simulation are *k:* gain, a T_1 : time constant and τ time delay.

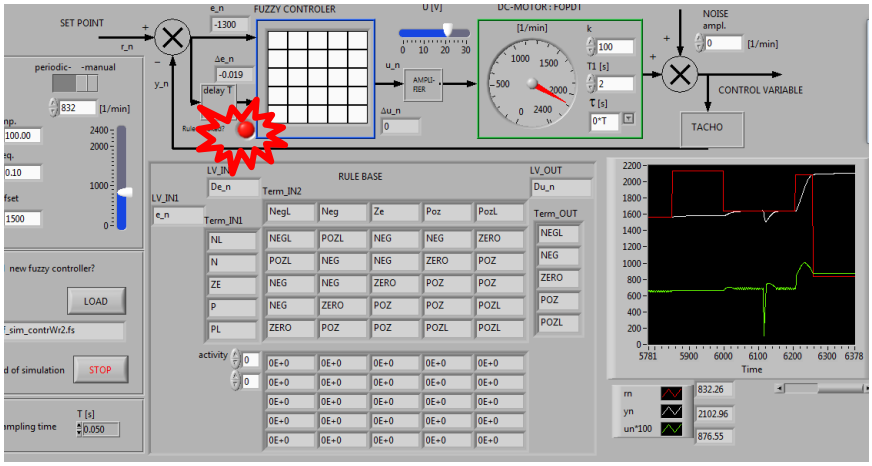
Simulation of Fuzzy PI-control



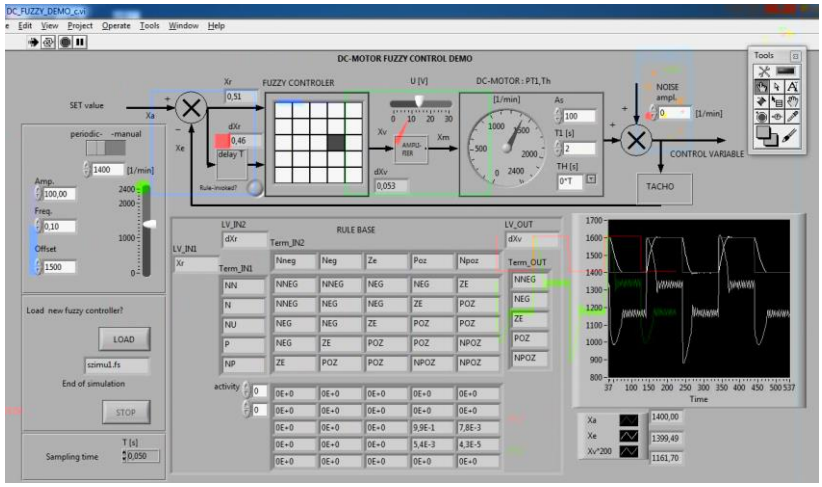
No active rule



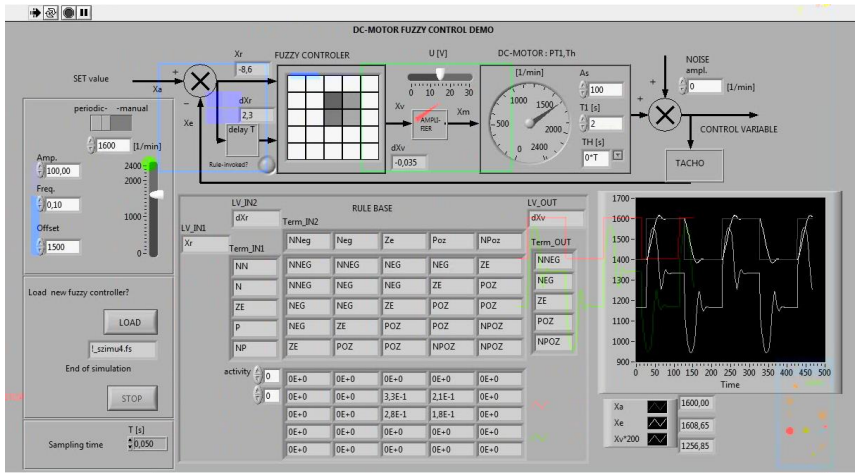
No active rule



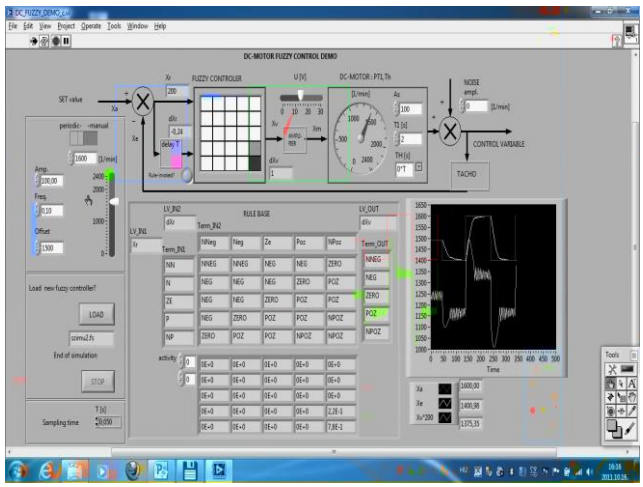
Fuzzy PI-controller: first probe



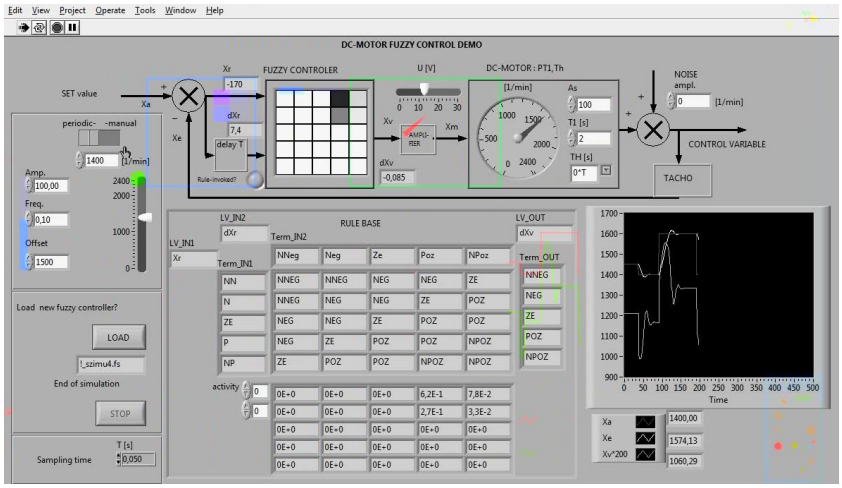
Fuzzy PI-control: *a*: revision, *b*: error in the rules



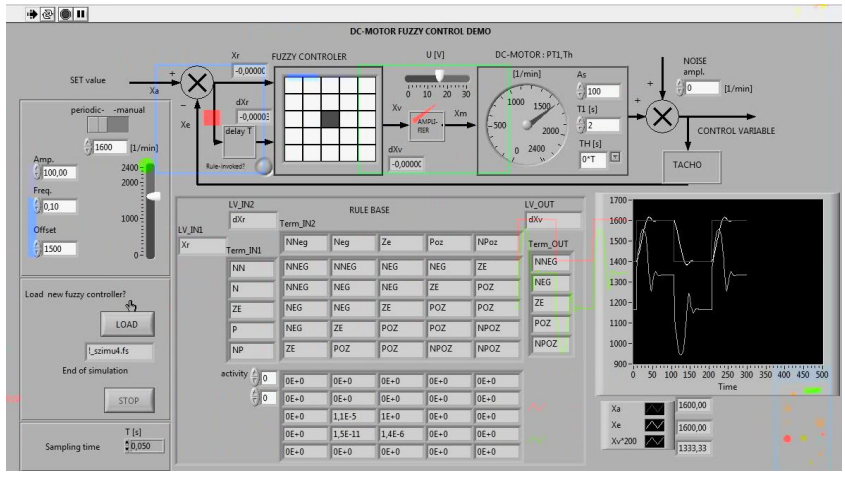
No active rule: the controller output will freeze
(Too large error or error-change: out of defined input limits)



The membership functions covers the possible input domains



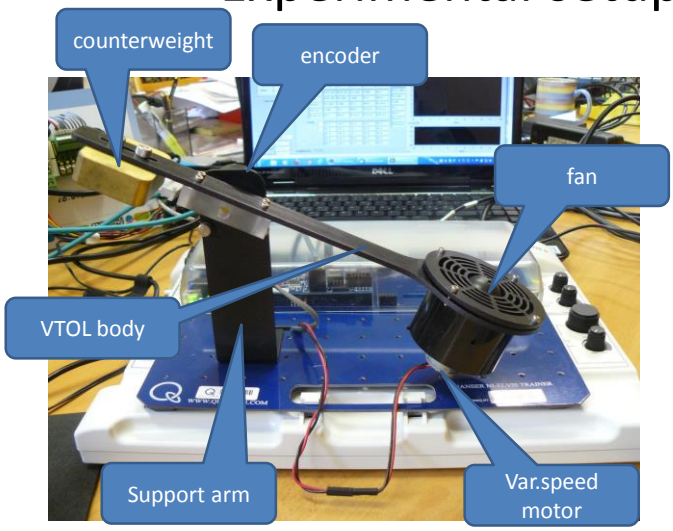
Smaller rule base



Intelligent control systems

SOFT CONTROL OF A 1 DEGREE-OF-FREEDOM HELICOPTER MODEL

Experimental setup



- The Quanser **Vertical Take-Off and Landing** (VTOL) Trainer is a balance like, **1 DoF helicopter model** The system consists of one variable-speed fan with safety guards, mounted on the end of a cantilevered arm.
- The VTOL device pivots about the pitch axis

VTOL model interfaced with a DAQ device

Measured Process Variables (PV):

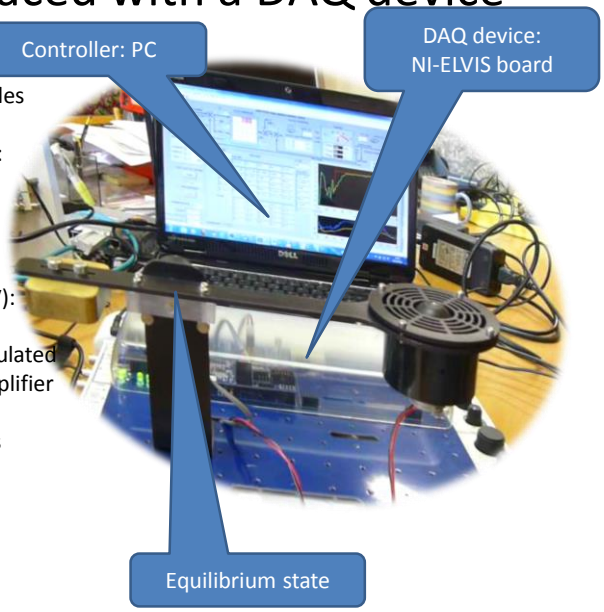
- Position measurement:
  - Optical encoder
- analogues variables:
  - Armature current

Manipulated variable (MV):

- motor voltage:
  - Pulse-Width Modulated (PWM) Power amplifier

The control algorithm was developed in:

- LabVIEW graphical programing system

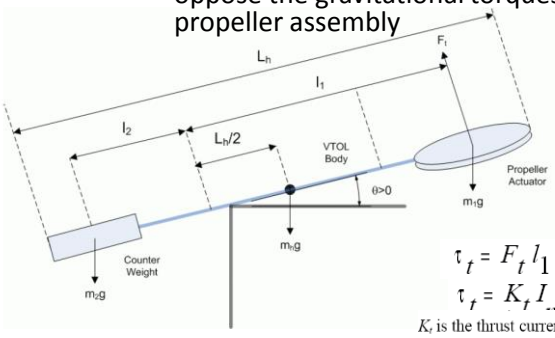


Variable speed of fan → change in pitch position

Change in motor voltage: change in armature current: change in torque: change in pitch position

Free-body diagram of 1DoF VTOL

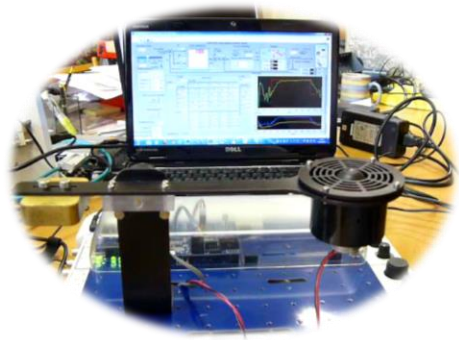
- The torque generated the propeller and the gravitational torque acting of the counter-weight act in the same direction and
- oppose the gravitational torques on the helicopter body and propeller assembly



$$\tau_t = F_t\, l_1$$

$$\tau_t = K_t\, I_m$$

K_t is the thrust current-torque constant



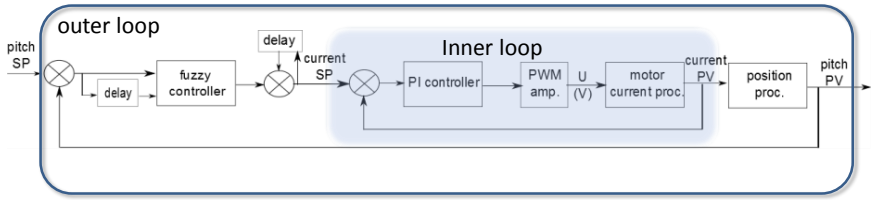
With respect to the current, the torque equation:

$$K_t\, I_m + m_2\, g \cos(\theta(t))\, l_2 - m_1\, g \cos(\theta(t))\, l_1 - \frac{1}{2} m_h\, g \cos(\theta(t))\, L_h = 0$$

Cascade control

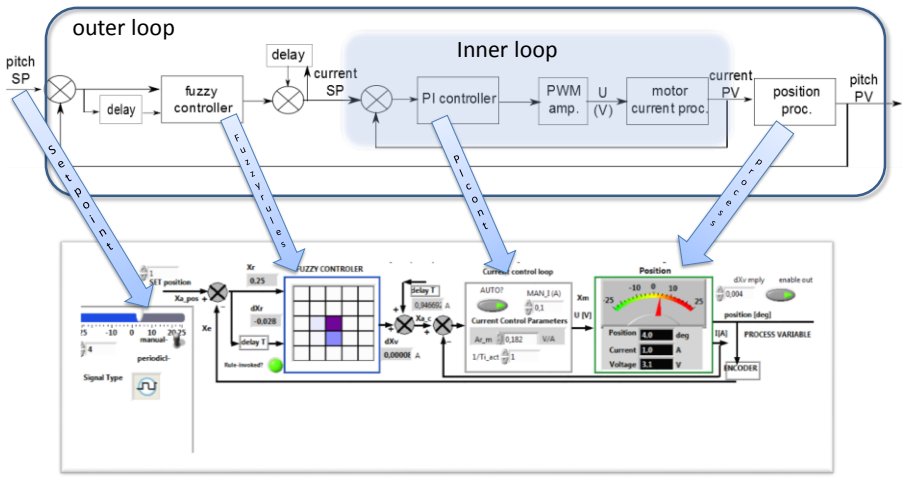
- The VTOL device is broken down into two subsystems:
  - the voltage-current dynamics of the motor
  - the current-position dynamics of the VTOL body
- Two different dynamic requres different control strategy:
- Cascade control:
  - Inner control loop: slave controller tuned for the voltage-current dynamics of the motor
  - Outer control loop: master controller tuned for the current-position dynamics of the VTOL body
- *This arrangement models the situation of a manual expert control*:
  a build-up, low level servo control (inner loop)
  and Human Operator supervisory control: (outher loop)
- Human Operator Type Control: Fuzzy Logic Controller as master controller in outher loop

Block diagram of the Cascade control

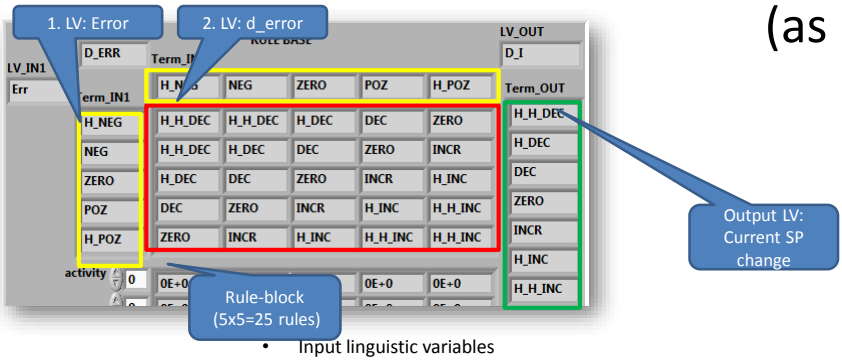


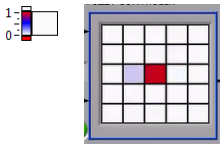
The cascade control consists of two nested control loops:
the ***inner loop*** with a PI current controller is designed to regulate the current inside the motor according to a desired current reference.
(Process Variable (PV) : I armature current ; manipulated variable (MV)): U motor voltage.
The current reference is genereated by the controller in ***outher-loop*** : a Fuzzy PI-controller
(Process Variable : the pitch of the VTOL trainer; output variable: Current Set Point (SP))

Cascade control loops on the LV front-panel



The fuzzy PI controller (as master)





- Input linguistic variables
  1. Error [$Err(iT)$=$SP_{pos}(iT) - PV_{pos}(iT)$] [5 terms]
  2. Delta Error[$D\_Err(iT)$=$Err(iT) - Err((i-1)T)$] [5 terms]
- Output linguistic variable: (manipulated variable (MV)).
  - Requred change in motor current [$D\_I(iT)$] [7 terms]
- *The rule-base is simmetric for a diagonal*
- Rule example: IF *Err = NN* AND *D_Err = NEG* THEN *dXv = Nneg*
- *Defuzzification method: CoA (Center of Area)*

Control requirements

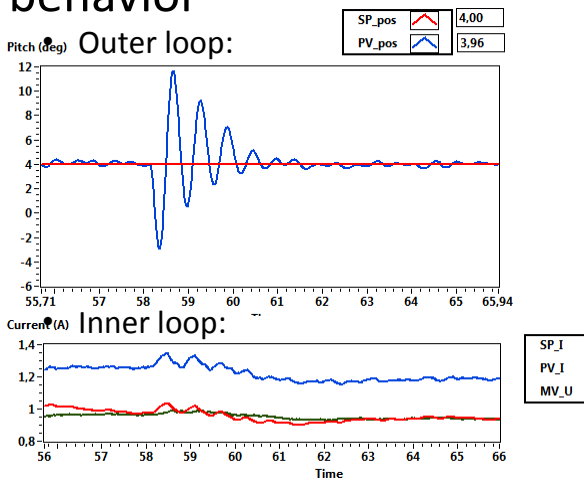
1. disturbance rejection – staying at a given setpoint and
2. command tracking – implementing setpoint changes

- Requirements in VTOL control*:*
  - *Avoide unstable states:* growing sinusoids are not allowed
  - *max 20% overshoot*
  - to reach the setpoint more quickly : low *rise time and settling time*
- Problem: the system is highly nonlinear and unstable!

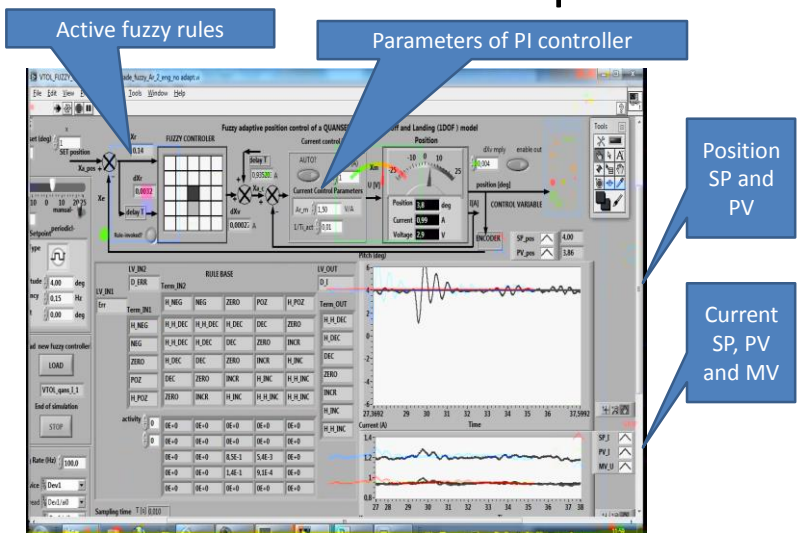
1. Problem: too slow command tracking or unstable behavior

- PI was tuned for disturbance rejection at +Pitch position
- Disturbance damped

(but unstable state can occure at neg. Pitch position!)

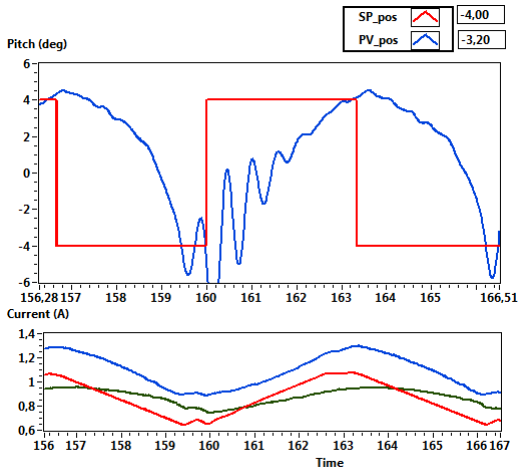


PI parameters tuned for + position

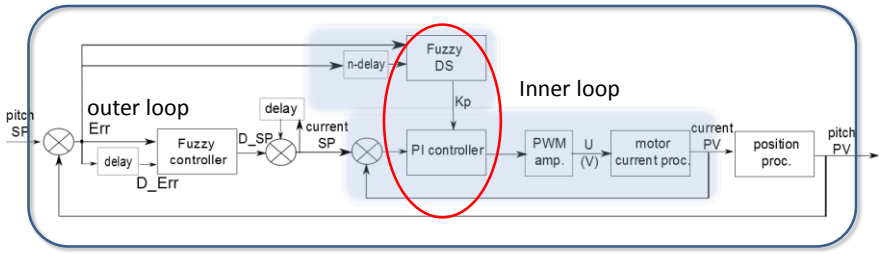


Attemps to tune for disturbance rejection at negative position

- Too slow control, no steady-state reaching in a desired time period!

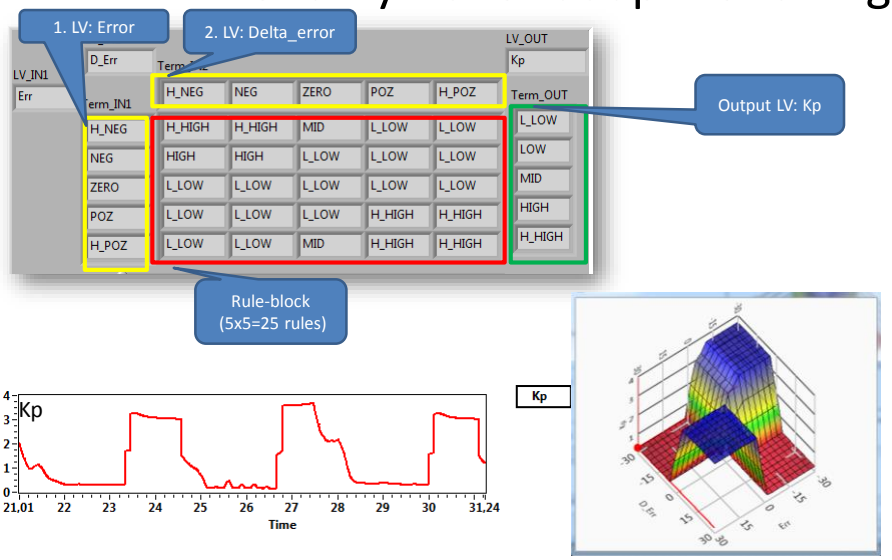


Solution: Fuzzy Decision System for Gain Sceduling Control



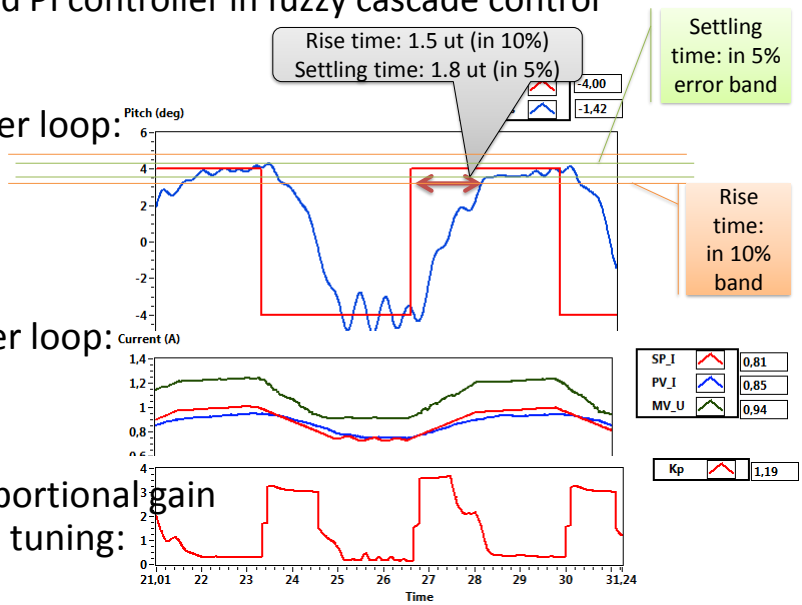
- K_p : Proportional gain, is tuned, Ti: integration time , constant
- $MV_k = MV_{k-1} + K_p\left((Err_k - Err_{k-1}) + \frac{\Delta T}{T_i} * Err_k\right)$ /Dig. PI-algorithm
  - ΔT: sampling interval, continous time=k ΔT
  - $Err_k = SP\_I_k - PV\_I_k$
- $K_i = K_p \frac{\Delta T}{T_i}$: Integral gain, increases with increasing K_p !

The fuzzy DS for adaptive tuning K_p



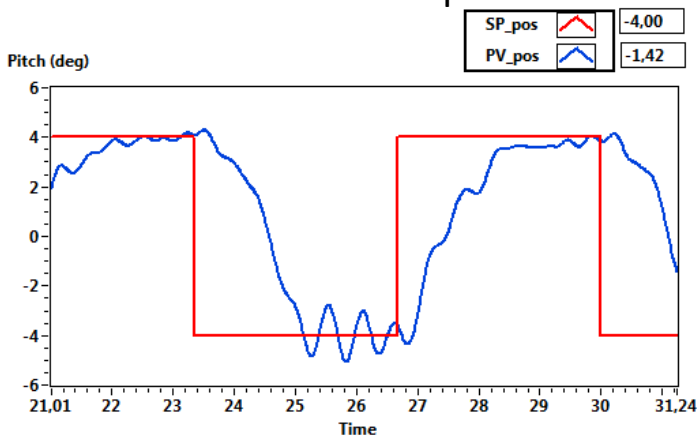
Adaptive tuned PI controller in fuzzy cascade control

- Outer loop:
- Inner loop:
- Proportional gain (Kp) tuning:

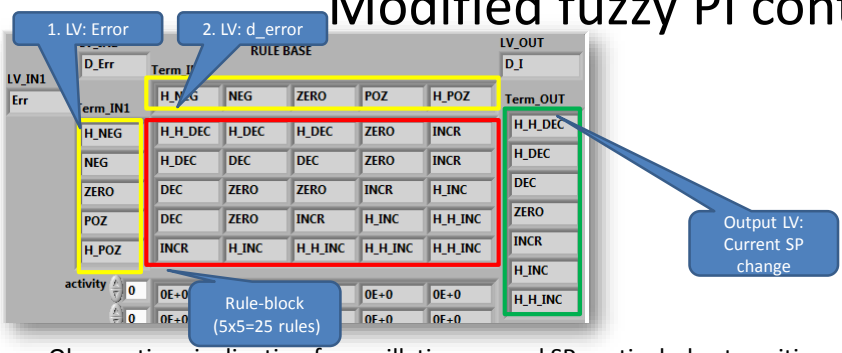


2. Problem : nonsimmetric dinamic characteristic of the VTOL model

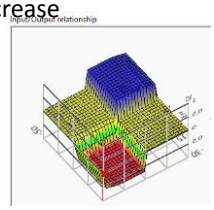
- Oscillation around the lower position

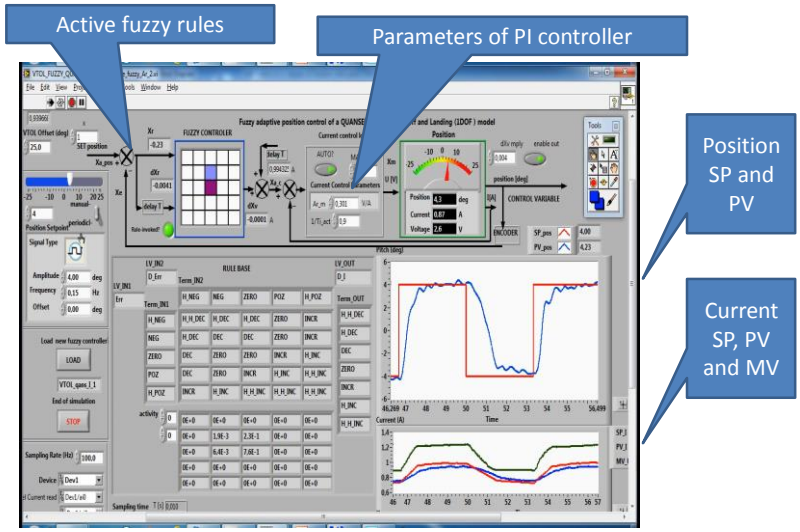


Modified fuzzy PI controller



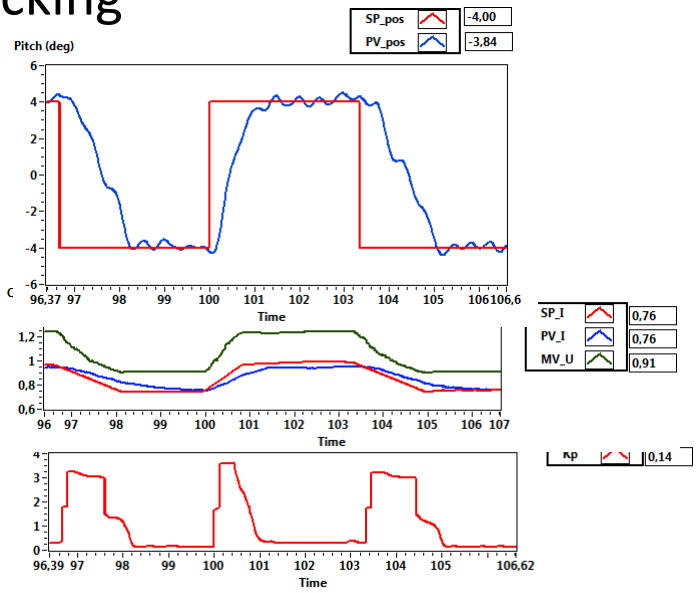
- Observation: inclination for oscillation around SP particularly at positions below of horizont. Even growing sinusoid can occure.
- Expert suggests: avoiding intensive decreasing , effort to increase
- Possible solutions:
  - LV terms membership function modification
  - and/or rule base modification, simmetry broken (in Figure)



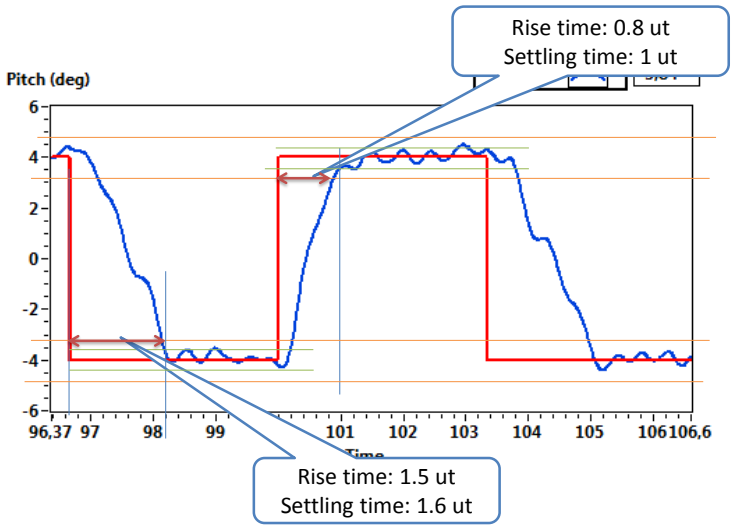


Command trecking

- Outer loop:
- Inner loop:
- Proportional gain (Kp) tuning:

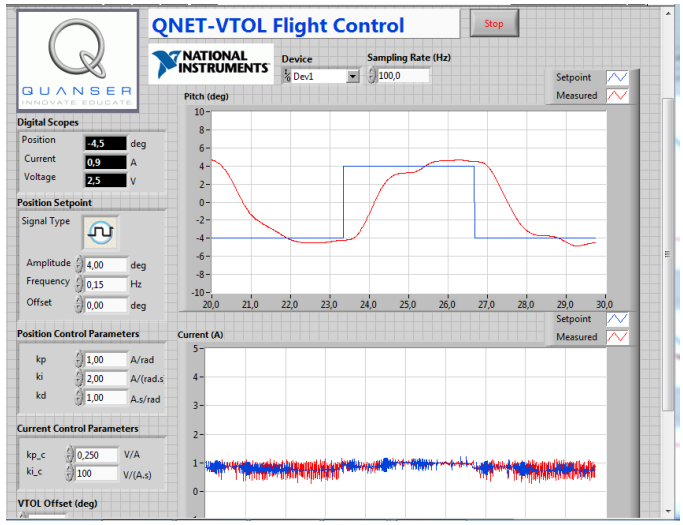


Results

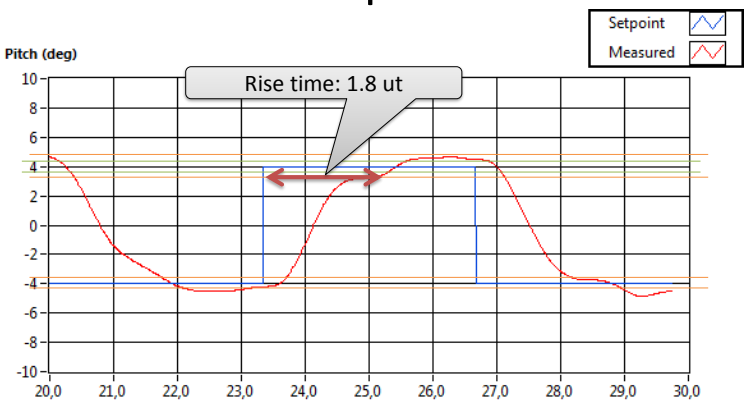


For reference: Cascade control with PID controller in the outer loop

- parameter settings by the manufacturer



Cascade control with PID controller in the outer loop



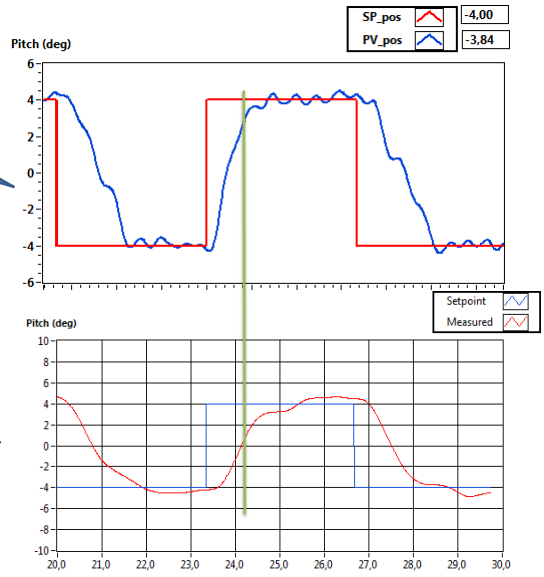
No settling time into 5%band !

Cascade control with Fuzzy controller in the outer loop, fuzzy adaptive PI controller in te inner loop

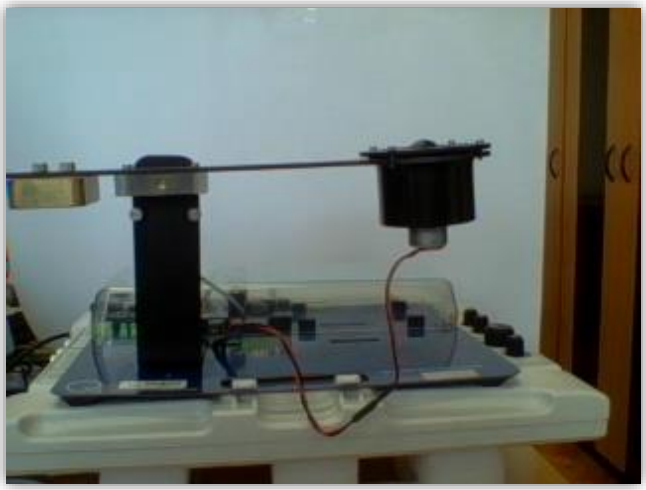
Rise time: 0.8 / 1.6 ut
Steady-state

Cascade control with PID controller in the outer loop

Rise time:1.8 ut
No steady-state
reach in desired
time-interval



Command trecking & disturbance rejection



Suggested textbooks

- R. C. Dorf, R. H. Bishop, Modern control systems, 12.ed. Prentice Hall, 2011.
- Control Systems Engineering, 8e
- L. A. Bryan, E. A. Bryan, PROGRAMMABLE CONTROLLERS, THEORY AND IMPLEMENTATION, An Industrial Text Company Publication Atlanta • Georgia • USA, 1997, ISBN 0-944107-32-X