Lab 2, Analysis and Design of PID Controllers

1 Goal

The main goal is to learn how to design a PID controller to handle reference tracking and disturbance rejection. You will design the controller and analyze its characteristics (settling time, stability, overshoot, steady-state error).

The PID controller is implemented in software, written with ISaGRAF which is a development environment for a Programmable Logic Control, PLC. Therefore, to learn how to program a PLC is also a goal of the lab.

2 The Process

The process to control is depicted in fig 1. It consists of two water tanks, the lower tank is filled from the upper tank, which in turn is filled by a pump. There is an outlet from the lower tank and, to introduce a disturbance, also the upper tank has an outlet. The measured value is the level of the lower tank.

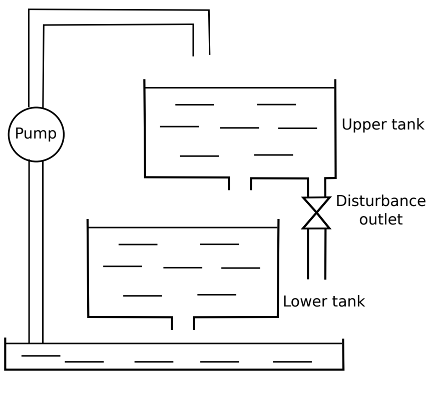


Figure 1: The controlled process

3 Introduction to PLC

IEC 1133-3 is an often used standard for Programmable Logic Control, PLC, control systems. This standard covers 5 languages, of which the latest are one graphic and one literary. The graphic is Sequence Function Chart SFC, which is similar to a flow chart, and the literary is Structured Text, ST, which is similar to most structured programming languages. In this lab we will use SFC to create a flow chart, and we will use ST for transitional conditions and outputs.

In PLC systems, the program written according to the IEC 1133-3 standard mentioned above is downloaded to a PLC computer with outputs for controlling actuators and inputs for sensor signals. Our development environment for these programs is called ISaGRAF and is installed on the computers in the lab.

The IEC 1133-3 standard is used both for automation and for control systems. The main focus of the lab is of course control systems, but as an introduction to PLC programming you will also write an automation program for pneumatic cylinders.

To be able to use ISaGRAF on the lab computers you must download the file called `isawin.zip` from the `Resources` page on the course website, `https://www.kth.se/social/page/formelsamling-2/` You must be logged in to KTH Social to be able to access this page. The `isawin.zip` file should be unpacked in the root of your home directory, that is `H:`

4 Preparation Tasks, to be solved *BEFORE* the lab

Task 1, Reading

- Read Chapter 11 in the course text book, and understand how a controller can be designed.
- Read Appendix 1 in this tutorial and try to understand the lab equipment.
- Read Section 5 in this tutorial and try to understand what to do during the lab.

Task 2, Understand Some Useful Hints

Following are some more facts that should be understood before the lab.

As explained in chapter 11 in the course text book, there are many different methods for controller design, but none of them is perfect. At the lab, we will try Lambda tuning (described in section 11.2 in the course text

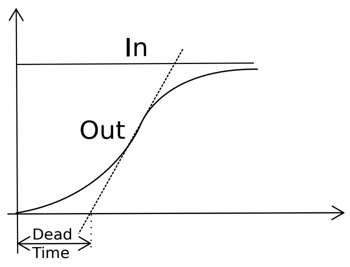


Figure 2: How to estimate dead time of a second-order system

book) and, if time allows, Ziegler-Nichols tuning (also described in section 11.2 in the course text book).

The main differences between these two methods are:

- Lambda tuning gives a system that has good stability margins and no overshoot, but is relatively slow.
- Ziegler-Nichols tuning gives a system that is fast but has poor stability margins and might also have considerable overshoot.
- Ziegler-Nichols tuning is more sensitive to error in the estimation of dead-time, especially if the dead time is small compared to the rise time, as is the case with our process. Also, the plot we use to measure the time constants of the process is produced using a sampling time of one second, which is far too long to get an accurate estimation of the dead time.

Some advices:

- Lambda tuning requires us to measure the dead time of the process, this can be performed as illustrated in figure 2. First, find the steepest part of the step response curve. Second, draw a tangent to this part of the curve. Last, measure the time between the intersection of the tangent with the x-axis and the point of the step in the input signal. This time is an approximation of the dead time.
- The values of K_P, T_i and T_d given by Lambda or Ziegler-Nichols tuning are estimates. The behavior of the system can most likely be improved by fine tuning these values.
- Ziegler-Nichols tuning requires estimation of amplitude margin and of ω_π. This can be quite time consuming using repeated step response measurements. A faster method is to estimate the time constants and amplification and then get A_m and ω_π from Matlab. The process

(water tanks) has the transfer function $G(s) = e^{-T_o s}\frac{K}{(1+sT_1)(1+sT_2)}$, which can be approximated to $G(s) = e^{-T_o s}\frac{K}{1+sT_1}$. Use the following matlab commands to calculate A_m and ω_π of the process.

```
G = tf([K], [T1, 1])
G.inputdelay = T0
margain(G)
```

Task 3, Controller Design an Analysis

Assume that the process has the transfer function, $G(s) = e^{-T_o s}\frac{K}{1+sT_1}$, where $T_0 = 0.1$, $T_1 = 25$ and $K = 0.5$

a) Design a PI controller using Lambda tuning.

b) Use Simulink to simulate the step response of the whole feedback loop (controller and process together). Try to change values of K_P and T_i and see how the system is affected.

c) Try to introduce a derivate, T_d, and see how the system is affected.

d) Design a PID controller using Ziegler-Nichols tuning. Again use Simulink to simulate the step response of the system and see how it is affected when changing the values of K_P, T_i and T_d.

Task 4, PLC Programming

a) Read Section 3, *Introduction to PLC* above.

b) To learn PLC programming with ISaGRAF, follow the ISaGRAF Quick Start Guide in Appendix 2. This task can only be performed in the lab room, since you need the pneumatic cylinders lab system. Contact one of the course teachers if you do not have access to the lab room.

5 Lab Tasks, to be solved at the lab

Task 1, Demonstrate your PLC Preparation Task

Demonstrate the program you created in *Preparation Task 4* to a teacher. This task may take up to 20 minutes and must not necessarily be done before you begin with the other lab tasks, but should be done in the beginning of the lab.

Task 2, System Identification

Remember that the maximum pump power is 10V, also remember that Amplifier Gain (see Appendix 1) should be set to 1x.

a) Set up the lab equipment as described in Appendix 1, but do not yet connect Smart I/O to the RCA breakout board. Instead use a DC power supply for pump power.

b) Set the pump power to approximately 6V DC and let the process stabilize. Then increase the pump power (about 1-2V) to generate an input step signal. Plot the the step response of the process using the scope meter and Fluke View as described in Appendix 1. Use the plot to calculate the dead time of the process as described in Preparation Task 2, also use the plot to calculate the time constant of the process. The amplification of the process can be found by calculating $K = \frac{U_{OUT}}{U_{IN}}$.

Task 3, Controller Design and Analysis

a) Use Lambda tuning to calculate K_P and T_i of a PI controller.

b) The controller is an ISaGRAF program called *pidcontr*, download the *pidcontr* program to Smart I/O. Before pressing the start button, enter the values of K_P and T_i that you calculated in the previous task, also set T_d to zero since there is no derivative in this controller. Also set the reference value (called *bor* in the program) to for example 5000. Let the system stabilize and then make a small change in the reference value, for example to 5200. Now you can plot the step response of the system using Fluke View, and you can also see the values of reference input, measured output and errors in ISaGRAF.

c) Calculate settling time, stability, overshoot and steady-state error of the process when using your controller and changing the reference value.

d) Calculate settling time, stability, overshoot and steady-state error of the process when using your controller and introducing a disturbance.

e) Try to improve the behavior of the system by changing values of K_P and T_i and by introducing a derivate, T_d. Note that this can be done without stopping the controller (the *pidcontr* program).

Task 4

If time allows, repeat Task 3 using Ziegler-Nichols tuning.

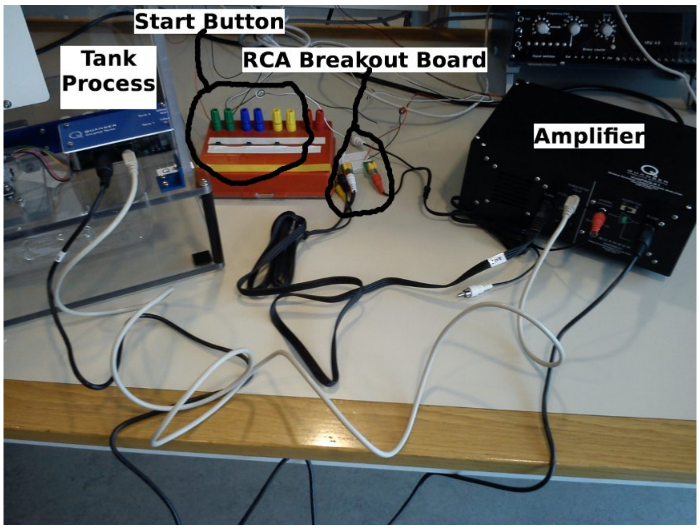


Figure 3: Water tank, start button, RCA breakout board and amplifier

6 Appendix 1, Lab Setup

6.1 Wiring

Figures 3 and 4 shows most components of the lab setup. Following is a wiring summary.

Sensor Output From Tank to Amplifier Use the light gray 6-pin-mini-DIN-to-6-pin-mini-DIN cable to connect the *Pressure Sensors Connector* of the tank process to the amplifier (VoltPAQ-X1) socket labeled *S1 & S2*.

Pump Power From Amplifier to Tank Use the 4-pin-DIN-to-6-pin-DIN to connect the tank process' *Pump Connector* to the amplifier socket labeled *To Load*.

Pump Power From RCA Breakout Board to Amplifier Use the 2xRCA-to-2xRCA cable to connect the socket labeled *From D/A* on the amplifier to one of the connectors on the RCA breakout board. In fig. 3, this is the rightmost connector on the RCA breakout board, that has a red RCA plug connected.

Sensor Output From Amplifier to RCA Breakout Board Use the 5-pin-DIN-to-4xRCA cable to connect the socket labeled *To A/D* on the amplifier to one of the connectors on the RCA breakout board. In fig. 3, this is the leftmost connector on the RCA breakout board, that has a white RCA plug connected. It is essential that the *white* RCA plug is used.

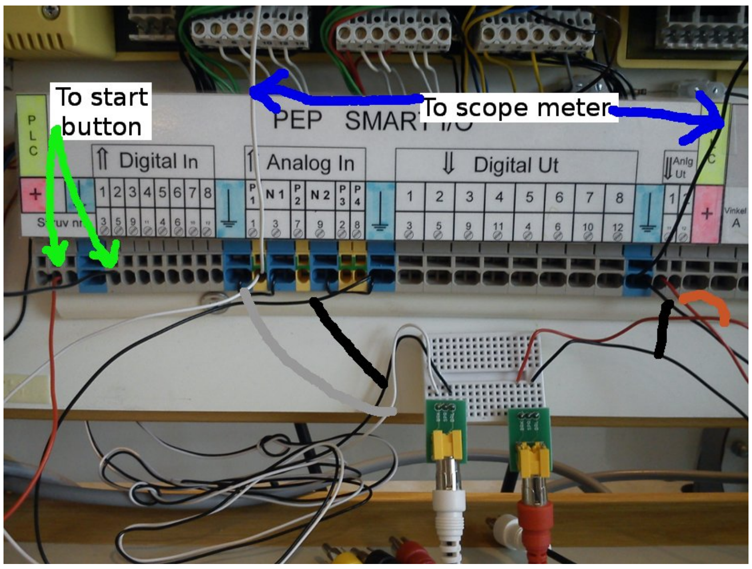


Figure 4: Smart I/O connections. Each of the gray, read and black lines connect two ends of the same cable, illustrating how the RCA breakout board is connected. The blue arrows show the cables that are connected the scope meter and the green arrows show the cables that are connected to the start button.

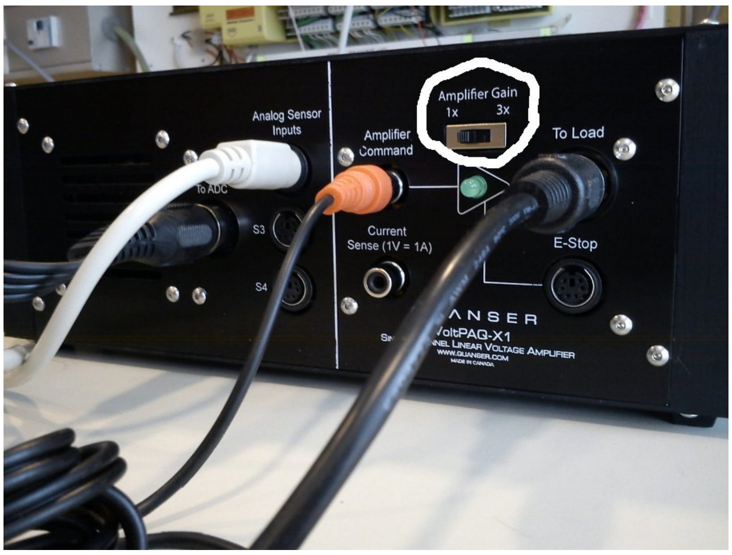


Figure 5: Amplifier Gain should be set to 1x.

Pump Power From Smart I/O to RCA Breakout Board The red and black cable connected to the red RCA plug in fig. 4 is the pump power connection from Smart I/O

Sensor Output From RCA Breakout Board to Smart I/O The white and black cable connected to the white RCA plug in fig. 4 is the sensor output connection to Smart I/O

Sensor Output to Scope Meter The blue arrows in fig. 4 show the white (signal) and black (ground) connections to the scope meter.

Amplifier Setting Note that *Amplifier Gain* should be set to 1x as illustrated i fig. 5.

Start Button The controller will start when the Start Button is pressed. Pressing the start button should connect *PLC+* to *Digital input 1*on Smart I/O, see figure 4.

6.2 Scope Meter and Fluke View

The *Fluke View* program is used to plot the measurements made with the scope meter. Note that Fluke View and ISaGRAF can not be executed on the same computer since they both use the computer's COM port. Following is a description of how to start Fluke View and make a plot.

1) Connect the signal you want to plot to channel A on the scope meter. Figure 4 shows where to find the output of the tank's pressure sensor.

2) Use the optocable to connect the scope meter to COM port A on the computer.

3) Set the scope meter to Probe 1:1 by first pressing the *LCD* button, then the *PROBE CAL* soft button, select 1:1 with the *up/down arrow* keys and finally press the *Enter* soft button.

4) Press the *Meter* button and then the *V DC* soft button on the scope meter.

5) Start *FlukeView ScopeMeter* on the computer, in the dialog choose COM1, 19200 baud and CONNECT.

6) The recording is started with the *V* Ω *Hz* button, in the dialog tic *DC* and un-tic *rms-AC*.

7) The recording is stopped with the *STOP* button.

7 Appendix 2, ISaGRAF Quick Start Guide

1) In the lab room, log in with your KTH account to the computer at one of the two desks with the pneumatic cylinders lab system. Contact one of the course teachers if you do not have access to the lab room.

2) If not done previously, download the file called `isawin.zip` from the `Resources` page on the course website, `https://www.kth.se/social/page/formelsamling-2/` You must be logged in to KTH Social to be able to access this page. The `isawin.zip` file should be unpacked in the root of your home directory, that is `H:`

3) Start `ISaGRAF PEP (V3.32) I0100` → `Projects`

4) First, we create a new project. Chose `File` → `New`, give your project a name, let's call it `projone` and click `OK`

5) Double click your project (`projone`). Now you can see a window showing all programs in your project, there are no programs yet so the work space is empty.

6) Chose `File` → `New`, give your program a name, let's call it `progone`. Chose `SFC` for `Language` and `Sequential` for `Style` and click `OK`.

7) Double click your program (`progone`)

8) What you now see is the editor where you write the program. The initial step (the box labeled 1) is showed. Now we create some more steps.

   (a) Click the area below the initial state.
   (b) Click the transition symbol (or press `F4`).
   (c) Click the step symbol (or press `F3`), now you have a flow of step 1, transition 1 and step 2.
   (d) Repeat step (b), then (c), then (b) again, now you also have transition 2, step 3 and transition 3.

9) Now we introduce a jump. Click the jump symbol (or press `F5`) and in the `Jump Destination` dialog mark `GS1` and click `OK`.

10) The program flow chart is ready, it is a continuous loop with three steps. Now it is time to declare variables, choose `File` → `Dictionary`.

11) Now you see the variable definition window, the work space is empty since there are no variables yet. To declare the boolean variable `START` you choose `Edit` → `New`. In the dialog enter `START` in the `Name` field and in the `Attributes` group choose `Input`. We choose `Input` here

since this variable will represent a hardware input, it will be tied to the signal from out start switch. Now click `Store` and you have the first variable.

12) In the same way declare the boolean input variables `B1`, `B2`, `F1` and `F2`. They are the switches at the end positions of the pneumatic cylinders.

13) Now declare the output boolean variables `C1` and `C2`, these signals will control the air valves of the two cylinders. Output signals are declared the same way as input signals, except that you choose `Output` in the `Attributes` group.

14) We are done declaring signals, close the dialog. Now we shall bind the signals to hardware ports. Chose `Tools` → `I/O connection`.

15) The `I/O Connection` window that is now displayed shows variable to port bindings, it is empty since we have not created any binding yet. To create a binding we first must define which hardware we use. Mark position 0 in the table and choose `Edit` → `Set board/equipment`. In the list, choose `sm_din1` which is our digital input module, then click `OK`. Now we have defined that we have a digital input module in position 0.

16) Now you can see the eight ports of the digital input module. Double click port 1, in the dialog mark the `START` signal and click `Connect`. Now the start signal is bound to port 1.

17) Repeat the previous step and bind `B1` to port 2, `B2` to port 3, `F1` to port 4 and `F2` to port 5.

18) Now we bind the output signals. First, mark position 1 in the table in the `I/O Connection` window. Then, choose `Edit` → `Set board/equipment`. In the list, choose `sm_dout1` which is our digital output module and click `OK`. Double click `logical_address` and set it to 2.

19) Now, bind the output signals the same way you bound the input signals in step 16. `C1` should be bound to port 1 and `C2` to port 2.

20) All variables are declared and bound. Close the `I/O Connection` window and the window with the variable declarations.

21) Next, we shall use the variables in our program. Go to the `SFC Program` window, which contains our flowchart.

22) Click the zoom symbol twice to make room for our statements. The zoom symbol looks like a blue lollipop.

23) Double click transition 1 and in the text area to the right enter `START and B1 and B2;` (don't forget the semicolon). This means our program will not pass from step 1 to step 2 until all signals START, B1 and B2 are `true`. When this happens, the cylinders are in their back positions and the start switch is activated. Again double click transition 1 so the text we entered appears in a yellow box next to transition 1.

24) Enter `F1;` at transition 2 in the same way you entered the condition for transition 1 in the previous bullet. This means the program will pass from step 2 to step 3 when F1 is active, which will happen when cylinder one hits the front switch. Then enter `B1;` at transition 3, can you understand when the program will pass this transition?

25) We are done defining transition conditions. Next, we shall define which output signals shall be active in which steps. Double click step 2 and press `N` in the tool-bar. In the text area to the right enter `C1 := true;` The complete text in the text area should now be:

```
ACTION (N):
   C1 := true;
END_ACTION;
```

Again double click step two to make the text appear in the blue box next to the step. Setting C1 to `true` puts the air valve of cylinder one in position to drive the cylinder forward.

26) Do the same for step three, except that here you should enter `C1 := false;` Setting C1 to `false` puts the air valve of cylinder one in position to drive the cylinder backward.

27) The program is complete, do you understand what it will do? Hint: cylinder two is not used.

28) Now it is time to verify and compile the program. Chose `File` → `Verify`, accept to save, enter a comment in the diary if you wish and click `OK`.

29) If you have entered everything correctly you should get a message that there were no errors. Accept to exit the code generator.

30) First we will simulate the program on the computer, without downloading to Smart I/O. First, close the `SFC Program` window, then, in the `Programs` window mark you program (`progone`) and choose `Debug` → `Simulate`.

31) In the `Debug programs` window double click your program and in the `SFC program` window zoom in so you can see the text for the steps and transitions in the program. Also choose `File` → `Dictionary`, which will display a window with all variables and their values.

32) The program is now running, to test it you should activate and deactivate input signals in the small window entitled `projone`, which contains a green list of input signals and a red list of output signals. You activate/deactivate inputs by clicking them in this window. You can choose `Options` → `Variable names` in this window to see the variable names. As the program executes, you can see variable values change and you can also see the execution advance through the steps (active steps become highlighted).

33) At last, we shall download the program to Smart I/O and steer the pneumatic cylinders. Close the running simulation. First we define which assembly code to generate, In the `Programs` window mark your program (`progone`) and choose `Make` → `Compiler options`, in the list mark `TIC code for Motorola` and click `Select` and `OK`.

34) Then, to compile, choose `Make` → `Make application`. Hopefully there was no error, accept to exit the code generator.

35) Now we define our connection to Smart I/O, choose `Debug` → `Link setup`. Set `Communication port` to `COM1` and click `Setup`. Set `Baudrate` to `9600`, `Parity` to `none`, `Format` to `8 bits, 1 stop` and `Flow control` to `hardware`. Click `OK` in both dialogs.

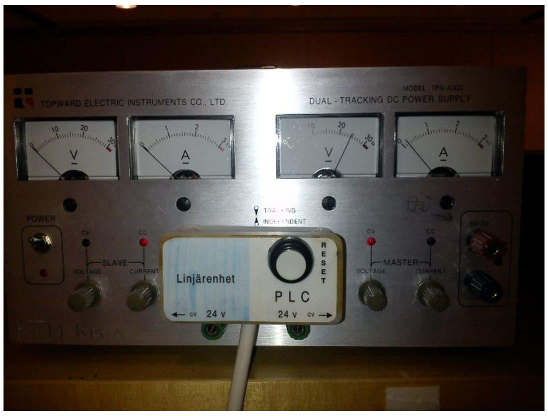
36) Check that PLC power is connected to the DC power supply and that the voltage is set to 24V, see figure 6. Chose `Debug` → `Debug`, now the `Debugger window` opens. If there is no old program left in Smart I/O the `Debugger` window says `No application`. If this is not the case you stop the old program by choosing `File` → `Stop application`.

37) To download the program, choose `File` → `Download`, mark `TIC code for Motorola` and click `Download`.

38) The program is now running, to see variable values on the screen you should double click your program in the `Debug programs` window and choose `File` → `Dictionary`, just as you did when you simulated the program.

39) Finally start by activation the start switch on the pneumatic cylinders training kit, remember to first turn on the air pressure, see figures 7 and 8.

40) Congratulations, you have completed your first PLC program. If you wish, you are very welcome to go on writing more programs.


TOPWARD ELECTRIC INSTRUMENTS CO., LTD
DUAL - TRACKING DC POWER SUPPLY
Linjärenhet
PLC
RESET
24 v
24 v
Figure 6: PLC power connected to DC power supply

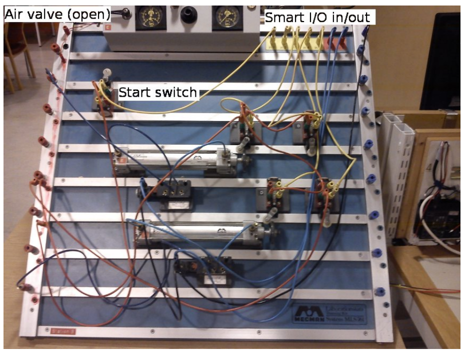


Figure 7: The pneumatic cylinders training kit

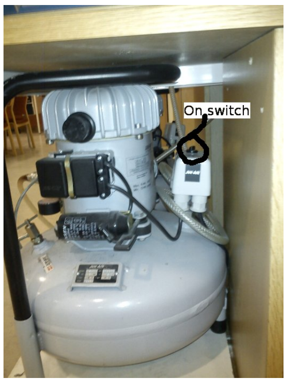


Figure 8: Compressor