

Lab 2, Analysis and Design of PID Controllers

IE1304, Control Theory

(with the 2015 rebuild equipment)

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

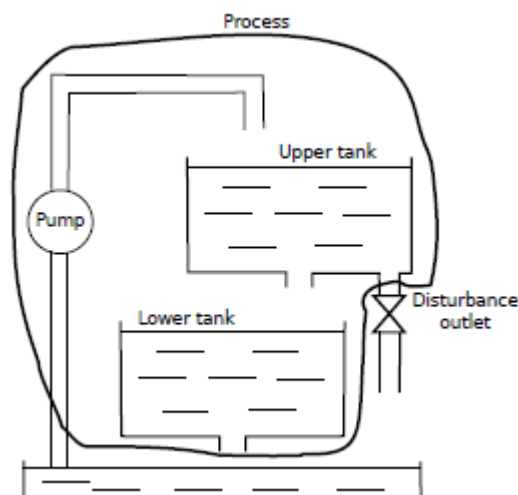


Figure 1: *Controlled process. Disturbance outlet and reservoir are not part of process.*

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. Note that disturbance outlet and reservoir below tanks are not part of the 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 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 a pushbutton and a 24V-lamp.

To be able to use ISaGRAF on the lab computers you must download the file called **isawin.zip** from the course website, <https://www.kth.se/social/course/IE1304/page/laborationer-2/>. You must be logged in to KTH Social to access this page. The **isawin.zip** file should be unpacked in the root of your home directory, **H:**

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

Task 1, Reading

- Read Chapter 11 in the course text book, understand how to design a controller.
- 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 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 Ziegler-Nichols tuning (described in section 11.2 in the course text book) and Lambda 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 less stability margins and might also have considerable overshoot.

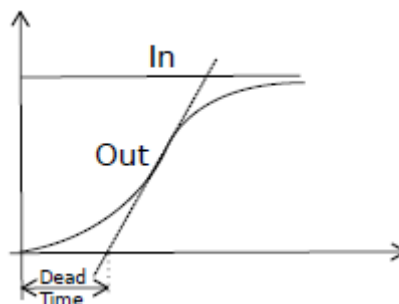


Figure 2: How to estimate dead time of a second-order system

- Lambda tuning requires us to measure the process dead time, this can be performed as illustrated in figure 2 and on page 197 in Thomas. 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 dead time. Remember to subtract this dead time when the time constant is calculated.

- Ziegler-Nichols tuning requires estimation of amplitude margin and ω_π . This can be quite time consuming using repeated step response measurements. Furthermore, a second order process without dead time, like ours, has infinite amplitude margin which makes such a measurement impossible. An alternative method is to approximate the process with a first order plus dead time model,

$$G_p(s) = e^{-Ls} \cdot \frac{K_p}{1 + sT_p}$$

Estimate the dead time and time constant as illustrated in Figure 2, also estimate the amplification and then get A_m and ω_π from Matlab. Use the following matlab commands to calculate A_m and ω_π .

```
GP = tf([KP], [TP, 1])
GP.inputdelay = L
margin(GP)
```

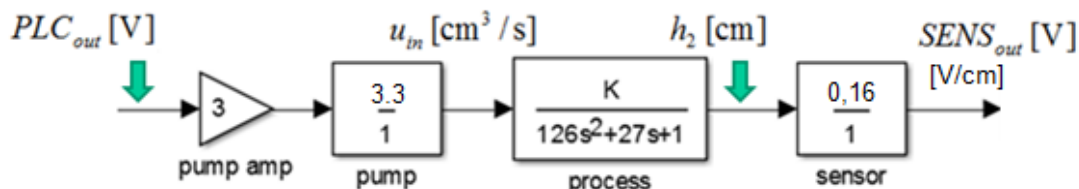
- The values of controller gain, K_R integration time, T_I and derivation time, T_D given by Lambda or Ziegler-Nichols tuning are estimates. The behavior of the system can most likely be improved by tuning these values.
- When simulating your controller in Simulink, you should model the process as the second order system it actually is,

$$G_p(s) = \frac{K_p}{(1 + T_1 s)(1 + T_2 s)}$$

Before the lab, when you have no measured values, you can set $K_p = 3 \cdot 3,3 \cdot 1,6 \cdot 0,16 = 2,5$ (pump amp, pump, process, sensor output). $T_1 = 6$ and $T_2 = 21$. These values are close to the actual process values.

Task 3, Process Transfer Function

The process is explained in section 2 above. Calculate transfer function *from pump amp voltage at Smart I/O output to lower tank depth*, using the following relations. Ignore disturbance outlet and reservoir, see figure 1. Both tanks have the same diameter $d = 4,5$ cm.



- Flow out of upper tank, u_1 [cm³/s] = $2,5 \cdot h_1$, where h_1 [cm] is the depth of the upper tank.
- Flow out of lower tank, u_2 [cm³/s] = $0,75 \cdot h_2$, where h_2 [cm] is the depth of the lower tank.
- Pump flow, u_{in} [cm³/s] = $3 \cdot 3,3 \cdot PLC_{out}$, where PLC_{out} [V] is the output voltage from Smart I/O.

Task 4, Controller Design and Analysis

Assume that the process has the transfer function, $G_p(s) = e^{-Ls} \cdot \frac{K_P}{1 + sT_P}$ where $L = 3s$,

$T_P = 25s$ and that total $K_P = 2,5$. These values are close to the tank model used during the lab.

a) Design a PID controller using Ziegler-Nichols tuning.

b) Use Simulink to simulate the step response of the whole feedback loop (controller and process together). Examine both process output and control signal. Remember to model the process as the second order system it actually is,

$$G_p(s) = \frac{K_P}{(1 + T_1 s)(1 + T_2 s)}$$

Assume that total $K_P = 3 \cdot 3 \cdot 1,6 \cdot 0,16 = 2,5$ (pump amp, pump, process, sensor output)

$T_1 = 6$ and $T_2 = 21$, which is close to the actual process values at lab.

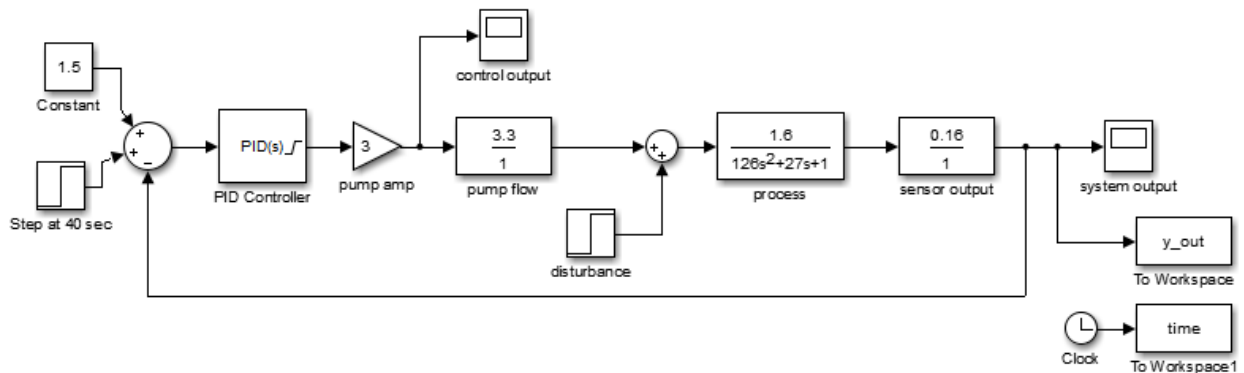


Figure 3: Simulation model used in Simulink

Use *PID Controller* block in Simulink, see figure 4. Set filter value, **N**, to 10 which is the value of the lab low pass filter. In the **PID Advanced** tab, set **Upper saturation limit** to 8 and **Lower saturation limit** to -8, since $3 \cdot 8 = 24V$ is max allowed pump voltage amplitude. Also set **Form** to **Ideal**, which is the formula used in the ISaGRAF controller at the lab. Finally, note that in this block you specify **integral gain**, I , not integration time, T_I . The relation is $I = 1/T_I$.

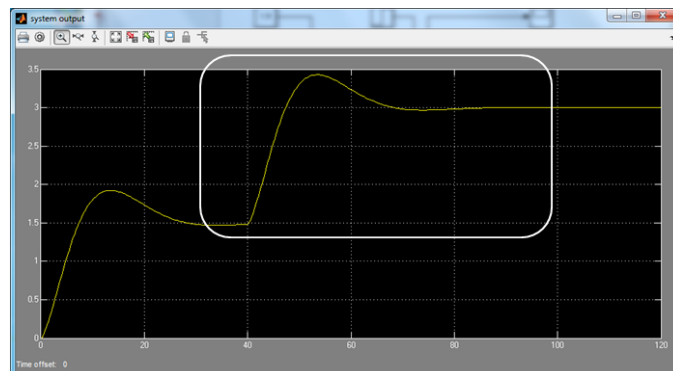


Figure 4: Step response in Simulink model. We are interested in the second step, from 1.5V to 3V

The scenario to investigate is a step from 1.5 to 3. To simulate this use the Constant block with a value of 1.5 and the Step block with Final value set to 1.5 and Step time to a time when



the system has stabilized at 1.5. Note that this will generate two steps, the first, from 0 to 1.5, is caused by the constant block and the second, from 1.5 to 3, is caused by the step function. It is the second step that is of interest to us.

Two *optional* **To Workspace** blocks can be used if you like to investigate the step response in Matlab (eg. use the **stepinfo** function – you then have to remove about 40 seconds of values from the **y_out** and **time** vectors with the Matlab colon operator before the actual step data).

Tune the controller by changing K_R , T_I and T_D until the following requirements are met:

- Rise time < 13s
- Settling time < 40s
- Overshoot < 10%
- Steady state error = 0
- Watch out for unjustifiably high control signals!

c) Design a PI controller using Lambda tuning. Again use Simulink to simulate the step response of the system and see how it is affected when changing the values of K_R and T_I . As in the previous task, you should model the process as a second order system. Also, you should examine both output and control signal. Tune the controller by changing K_R and T_I until the following requirements are met:

- Rise time < 35s
- Settling time < 45s
- Overshoot = 0%
- Steady state error = 0
- Watch out for unjustifiably high control signals!

Task 5, 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 SmartI/O PLC system (but you don't need the complete lab equipment with the water tanks). 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

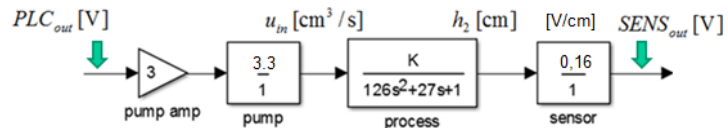
Task 1, Demonstrate your PLC Preparation Task

Demonstrate the program you created in Preparation Task 4 to a teacher.

Task 2, System Identification K L T_p

a) Set up the lab equipment as described in Appendix 1. (If this is not done in advance).

b) To get the step response of the process, download the ISaGRAF program **stepresp** to SmartI/O. Pressing the start button makes the program fill the tanks.



Wait for the lower tank level to stabilize at about 6 cm. Again press the start button. Every time the start button is pressed, the level will now alternate, between 6 cm and 27 cm, and a “short” pulse will indicate the step start time. The step begins on the pulse's negative slope. (This “indicator pulse” will not reach the process). Plot the step response of the process using an oscilloscope with a roll-mode timescale option as described in Appendix 1.

The “total” amplification of the process can be found by calculating $K = \frac{\Delta SENS_{OUT}}{\Delta PLC_{OUT}}$.

Measure the step input, the voltage at pump amplifier input PLC_{OUT} [V] with a voltmeter.

Measure the step response, the sensor voltage $SENS_{OUT}$ [V] with the oscilloscope.

Use the plot to measure the “dead time” of the process as described in Preparation Task 2, also use the plot to measure the time constant of the process. $L = ?$ s. $T_p = ?$ s.

Note that K is another amplification value than the previous calculated in Task 3, here the output is the sensor voltage, *not* the lower tank depth (≈ 0.16 V/cm – can be trimmed).

Task 3, Controller Design and Analysis

a) Use Ziegler-Nichols tuning and Matlab to calculate K_R , T_I and T_D of a PID controller.

b) The controller is an ISaGRAF program called **pidzn**, download it to Smart I/O. Before pressing the start button, enter the values of K_R , T_I and T_D that you calculated in the previous task. Also set the reference value (called **ref** in the program) to 10, which means a water level in the lower tank about 10 cm. (PLC reference has an appropriate scale factor). Let the system stabilize and then change the reference value to 20, plot the step response using the oscilloscope (roll-mode). Reference input, measured output, error and control signals can be seen in ISaGRAF.

c) Tune the controller by changing K_R , T_I and T_D until the system meets the requirements specified in preparation task 3. Note that this can be done without stopping the **pidzn** program.

To avoid errors due to integration windup, clear integration error by pressing the start button whenever you change control parameters. Then let run for some time to stabilize before changing the reference value to generate a step. Turn off the amplifier when not used.

Task 4

If time allows, repeat Task 3 using Lambda tuning. Use the **pilambd** program instead of **pidzn**.

6 Appendix 1, Lab Setup

6.1 Wiring

Following is a wiring summary. (To use in case the lab equipment is not wired in advance).



Figure 5: Water tank, amplifier and PLC

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*.

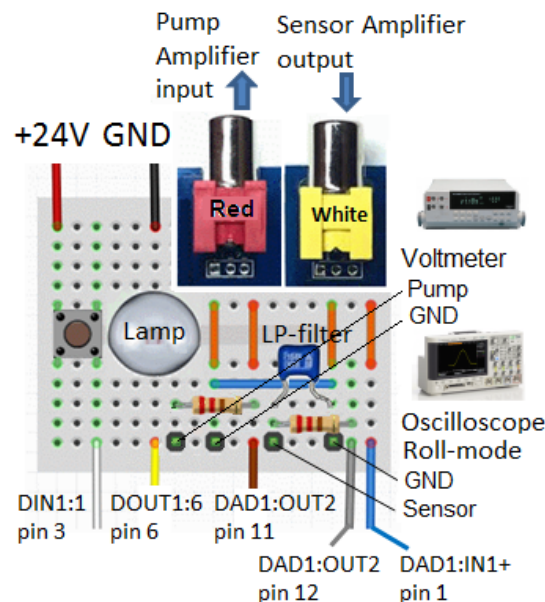


Figure 6: Breadboard for RCA contacts and PLC connections, start button, and 24V lamp.

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. 5, this is the 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. 5, this is the one on the RCA breakout board, that has a white RCA plug connected. It is essential that the white RCA plug is used.

Sensor Output to Oscilloscope There are two moxex pins on the breakout board for connections to the oscilloscope. Two other moxex pin are for monitoring the voltage to the pump amplifier with a voltmeter.

Amplifier Setting Note that *Amplifier Gain* should be set to **3x** as illustrated in fig. 7.

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 6.



Figure 7: Amplifier Gain should be set to 3x.

6.2 Chart recorder, oscilloscope with roll-mode

In order to see the output from the level-sensor, which will be a slow transient, we should need a chart-recorder. At lab we are using a digital oscilloscope that has a setting for roll-mode, which mimics the behaviour of such a chart-recorder.

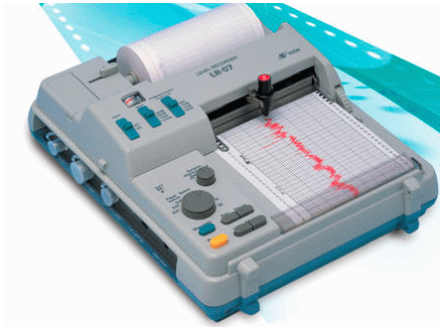
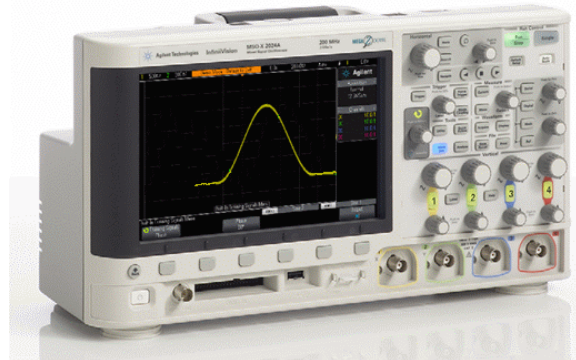


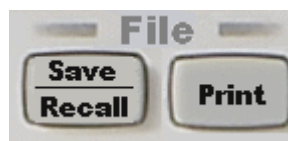
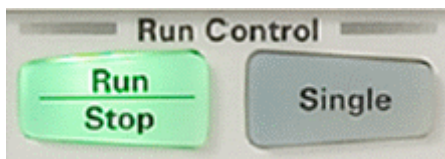
Chart-recorder (expensive)



Digital oscilloscope (as we already have them)



Press **Horiz** button. Press Softkey **Time Mode** select **Roll** with the Entry-knob. Start with Horizontal setting **10.00 s/div**, and Vertical setting **500 mV/div**. (Probably you are using a 10:1 probe – Choose Vertical **1** and Softkey **Probe** to set the damping to **Ratio 10.0:1**).



With the **Single** button at the Run Control, you can **Stop/Clear-Restart** the recording. With the **Cursors** button and the Softkey **Cursors** you can then select one cursor (y_1 , y_2 , x_1 , x_2) at a time to move with the cursor knob. Both absolute and differential values (Δ) between the cursor positions will be shown.

(File-menu Save/Recall)

You can store the screen-image on an USB-stick, in different formats. Either as an image or as a textfile with numerical values. The numerical values can be used in Excel or in Matlab. See the oscilloscope manual for details if you are interested in this option.

7 Appendix 2, ISaGRAF Quick Start Guide

1) In the lab room, log in with your KTH account to a computer with the PLC equipment. 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 row 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 our start switch. Now click Store and you have the first variable.

12) Now declare the output boolean variable **LAMP**, which is for the 24V lamp we are going to control. Output signals are declared the same way as input signals, except that you choose Output in the Attributes group.

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



14) 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.

15) Now you can see the eight ports of the digital input module. Double click **port1**, in the dialog mark the **START** signal and click **Connect**. Now the start signal is bound to **port1**.

16) Now we bind the output signal. 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.

17) Now, bind the output signal the same way you bound the input signals in step 15. **LAMP** should be bound to port 6.

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

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

20) Click the zoom symbol twice to make room for our statements. The zoom symbol looks like a blue magnifying glass.

21) Double click transition 1 and in the text area to the right enter **START;** (don't forget the semicolon). This means our program will not pass from step 1 to step 2 until the start button is pressed, (when **START** is true). Again double click transition 1 so the text we entered appears in a yellow box next to transition 1.

22) Enter **GS2.T>=t#1s;** 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 a timedelay of 1 second has elapsed. Then enter **GS3.T>=#t1s;** at transition 3. Now when the program pass from step 3 to step 1 there is also an 1 second delay.

23) We are done defining transition conditions. Next, we shall define which output signals shall be active in which steps. Double click step 2 and in the text area to the right enter **LAMP(S);** which means that variable **LAMP** is Set (lamp is on). Again double click step two to make the text appear in the blue box next to the step.

24) Do the same for step 3, except that here you should enter **LAMP(R);** this means that variable **LAMP** is Reset (lamp is off).

25) The program is complete, do you understand what it will do?

26) 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**.

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

28) 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**.



29) 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.

30) 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).

31) At last, we shall download the program to Smart I/O and steer the 24V lamp. 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**.

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

33) Now we define our connection to Smart I/O, choose **Debug → Link setup**. Set Communication port to **COM1** and click **Setup**. (Be prepared that some of the lab computers could use **COM2** instead). Set Baudrate to 9600, Parity to none, Format to 8 bits, 1 stop bit and Flow control to hardware. Click **OK** in both dialogs.

34) Check that PLC power is connected to a DC power supply and that the voltage is set to 24V (18-36V) and that the polarity is correct. **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**.

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

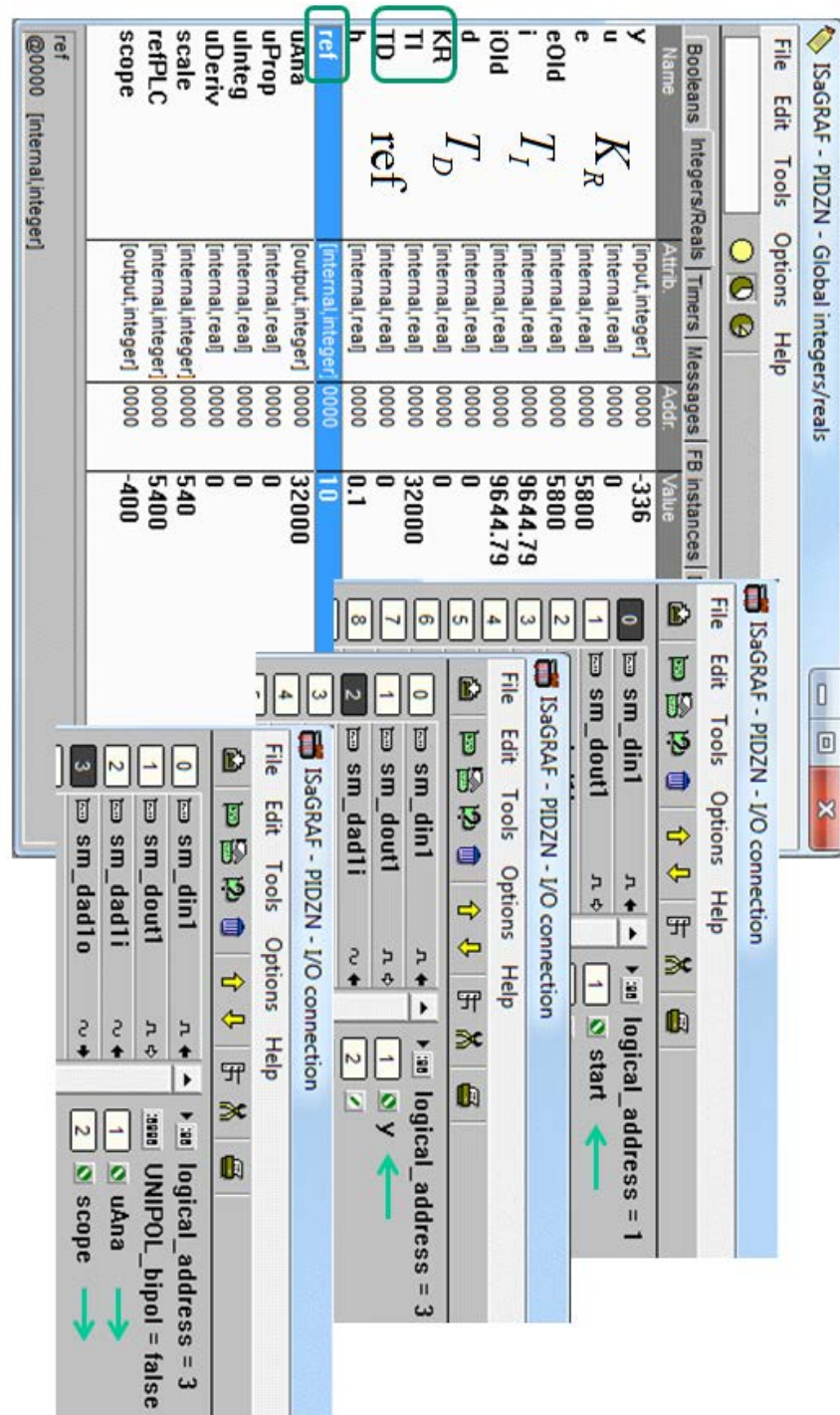
36) The program is now running, to see variable values on the screen you should double click your program in the Debug programs window and choose **File → Dictionary**, just as you did when you simulated the program.

37) Finally light the lamp by pressing the button (on the breadboard). As long as you press the button the lamp should blink.

38) Congratulations, you have completed your first PLC program. If you wish, you are very welcome to go on writing more programs! Can you toggle the lamp from the switch?

You can inspect all programs that we use at lab by downloading images of their graphs from the courseweb. You don't need to do any more ISaGRAF programming after this, as all further programs we use will be included at lab.

8 Appendix 3, ISaGRAF Globals, I/O-connections



The screenshot displays the ISaGRAF software interface, specifically the Globals and I/O connections windows.

Globals Window: This window lists various global parameters and their attributes. The parameters are organized into columns: Name, Attribute, Address, and Value.

Name	Attribute	Addr.	Value
y	[input, integer]	0000	-336
u	[internal, real]	0000	0
e	[internal, real]	0000	5800
eold	[internal, real]	0000	5800
i	[internal, real]	0000	9644.79
id	[internal, real]	0000	9644.79
d	[internal, real]	0000	0
K _R	[internal, real]	0000	0
T _I	[internal, real]	0000	32000
T _D	[internal, real]	0000	0
ref	[internal, real]	0000	0.1
uana	[output, integer]	0000	32000
uProp	[internal, real]	0000	0
uInteg	[internal, real]	0000	0
uDeriv	[internal, real]	0000	0
scale	[internal, integer]	0000	540
refPLC	[internal, integer]	0000	5400
scope	[output, integer]	0000	-400

I/O connections Window: This window shows the connections for the ISaGRAF - PIDZN - I/O connection. It includes fields for logical addresses and connection status.

Connection	Logical Address	Status
sm_din1	1	start
sm_dout1	1	start
sm_dad1i	2	start

The screenshot also shows the UNIPOL_bipol = false setting and the scope connection status.

