Process Control

Laboratory Exercise X

Implementation of a Batch Process Control System

Department of Automatic Control
Lund University
Last updated March 2010

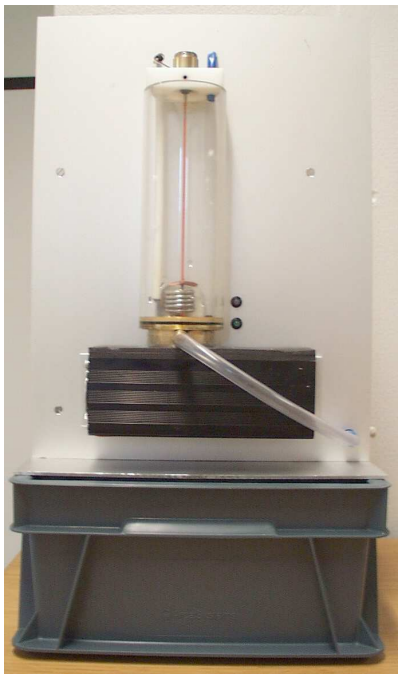


Figure 1 The batch reactor.

1. Getting started

Login with `lab_batch`, no password. To start both JGrafchart and the simulation write `start` in a terminal window (which you open by clicking the terminal icon in the menu bar.) You may close the terminal window once you have run the `start`.

Rather than starting from scratch, a JGrafchart file with some definitions and structure is available in `xml/lab3_start.xml`. it is opened using the 'Open' under the 'File' menu or hitting `Ctrl-O` when focus is on the `JGrafchart` window. (Double click on the `xml` directory icon and then on `lab3_start.xml` By now, you should have the following windows, See Figure 2:

- `JGrafchart` showing the `Top:Lab3` workspace
- `Plotter`
- `Tank Animation`
- `Client OpCom`

If any of them should crash, use the following: To start JGrafchart write `>JGrafchart`. To start the others write `>start_sim` in a terminal window.

Finally, ensure that the process is connected to a 230 V socket and to the lab PC via serial cable. There is a switch on the side of the process. Make sure it is switched on. A led at the front of the process is lit when the process is on. If the LED is green everything is OK. If it is red, press the reset button on the side of the process. If the LED still does not turn green, contact the lab assistant.

2. Introduction

In this laboratory exercise you will implement a sequential control system for a batch reactor process, see Figure 1. You will also implement a discrete PI controller for controlling the temperature of the reactor. The development of the control system will be made in JGrafchart, a graphical programming language implemented in Java and developed at the department. The control system will first be tested against a simulation and then evaluated against the real process.

The process to be controlled is a batch reactor, which is to be run in the following way: First an amount of reactant A is added with the use of a pump. Then the reactor is heated and an endothermic reaction starts, which turns A into the product B. When the reaction is ready the reactor is emptied using another pump. Once the reactor is empty it needs to be cleaned before the next batch can be made. In the laboratory exercise water is used as the reactant and product. An electric cooler simulates the endothermic reaction.

Preparations

Before the laboratory exercise you should have read this manual and solved the *preparatory* exercises. Make sure to study and understand the introduction to JGrafchart in Section A, prior to attending the lab.

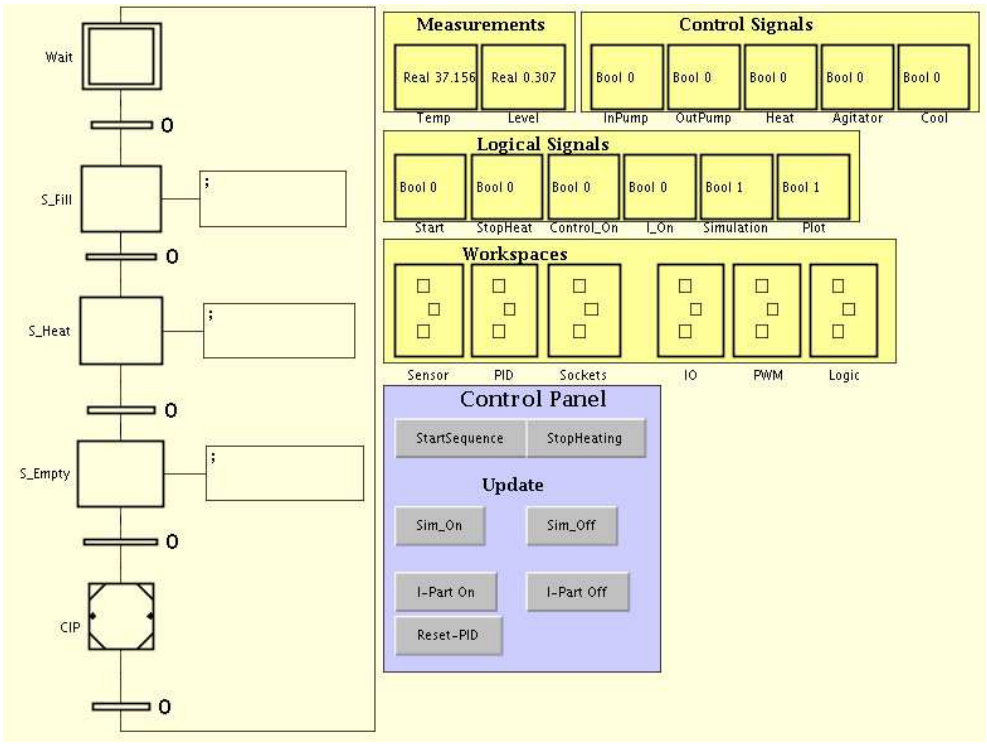
3. Laboratory Equipment

The process consists of a small tank with an electrical heater. It has a number of measurement and control signals, which are tied to variables in the `Top` workspace of the `Lab3` XML-file opened in JGrafchart.

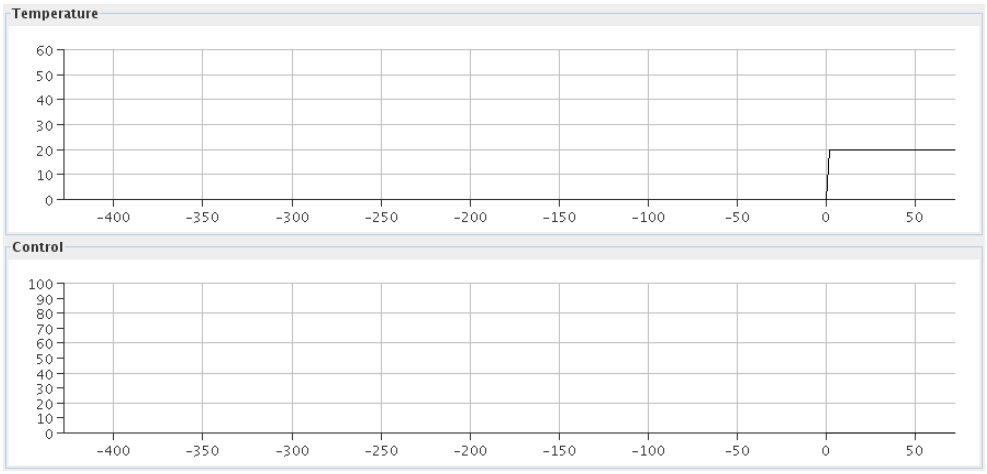
The electrical heater is on when the boolean variable `Heat` is 1 (true). In the bottom of the tank there is a cooler, which is used to simulate the reaction. The cooling is on when the boolean variable `Cool` is 1. An agitator in the tank makes sure there are no temperature gradients in the liquid. The agitator is started using the boolean variable `Agitator`. There is one pump for filling the tank and one for emptying the tank. They are controlled using the variables `InPump` and `OutPump`.

It is also possible to start or stop the pump, cooler and agitator from the `Client Opcom` window shown in Figure 2(d).

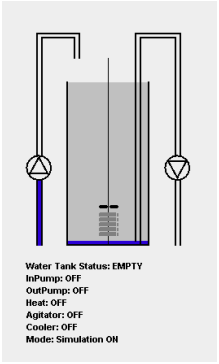
There are two real variables connected to transmitters in the process. The `Level` variable is in the range 0–1, corresponding to 0–10 V from the level sensor (larger number corresponds to higher water level). `Temp` is in the range 0–100, which corresponds to 0–100 °C.



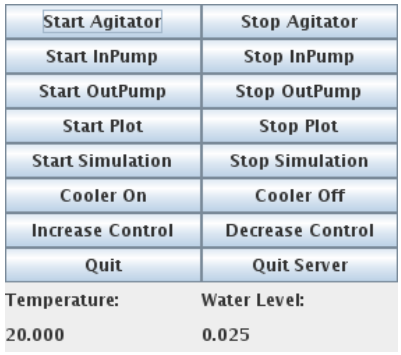
(a) JGrafchart with `Top: Lab3` workspace



(b) Plotter



(c) Tank Animation



(d) Client OpCom

Figure 2 Windows of the graphical user interface.

The process has some interlocks built in, e.g. it is not possible to heat the tank if the level is too low. In the case something prohibited takes place the process sets the `Error` signal variable to 1 and the LED on the front of the process turns red. If this happens, try to resolve the cause and then press the reset button at the side of the process. Ask the lab assistant if this fails.

4. Modeling and Simulation

We will control two states in the batch reactor: the water level and the temperature of the water. The water level dynamics is the simplest to model.

4.1 Water Level Dynamics

Let us denote the water level as a function of time with $h(t)$ in meters. The differential equation is then

$$A\frac{dh}{dt} = q_{in} - q_{out} \quad [\mathrm{m}^3/\mathrm{s}], \tag{1}$$

as the mass is conserved. Here q_{in} $[\mathrm{m}^3/\mathrm{s}]$ is the flow of water from the in-pump and q_{out} $[\mathrm{m}^3/\mathrm{s}]$ is the flow from the out-pump. The tank cross-section area is $A = 0.06^2\pi$ m^2. As the dynamics are simple there will be no need for advanced control, simple on/off control of the pumps will suffice. The pumps have a maximum capacity of 15 l/min, but this will not be fully exploited. The water level measurement from the process is normalized, $h(t)/h_{max}$, so that it becomes a number between 0 and 1 in JGrafchart.

4.2 Water Temperature Dynamics

Let us denote the temperature with $T(t)$ $[^\circ\mathrm{C}]$. In our simple model we will assume the temperature is uniform throughout the tank. This is not completely true, but we will use an agitator to make this assumption close to reality. The temperature dynamics are modeled by energy balance, described below. The combined specific heat of the water and the tank are represented by the constant c $[\mathrm{J}/\mathrm{kg}^\circ\mathrm{C}]$ and their combined mass is denoted m [kg]. The heat conduction between the tank and room is modeled by k $[\mathrm{W}/^\circ\mathrm{C}]$. Further, there is a heat source q_{heat} [W] and a heat sink q_{cool} [W]. The balance equation becomes

$$mc\frac{dT}{dt} = k\cdot(T_{room} - T) + q_{heat} - q_{cool}(T_{room} - T) \quad [\mathrm{W}]. \tag{2}$$

The room temperature T_{room} is approximately 20°C. Notice that if the heater and cooling is off the water temperature T will tend to T_{room}, which makes sense. (2) is only valid for temperatures below the boiling point and above the freezing point.

The cooling $q_{cool}(T)$ is provided by a *Peltier* thermoelectric element. Its efficiency depends on the temperature difference between the water and the room. It gets more efficient as the temperature of the water increases. When the temperature difference is zero, the cooling is about 40 W. In the working temperature range, the dependence is well described by a linear relation. The cooling is included in the lab to simulate the endothermic chemical reactions.

The heater can deliver a maximum of $q_{heat} = 150$ W. It will be scaled according to

$$q_{heat} = 150\frac{u}{100} \quad [\mathrm{W}],$$

where u is our control and is a dimensionless number between 0 and 100. u is the output from the controller we will design in JGrafchart.

As seen in (2) there are many physical parameters to determine. However, for our purpose there is no need to measure each single parameter. Instead, as we know the structure of the model, we can fit the model to some experimental data and get a reasonable result. Step response experiments yield

$$\tau \cdot \dot{T} + T = \kappa_1 + \kappa_2 \cdot u \quad [^\circ\mathrm{C}] \tag{3}$$

when the cooling is on, at normal room temperature, and the level is approximately one cm over the agitator. The open-loop time-constant τ is 1075 seconds. The other constants are $\kappa_1 = -4.33^\circ$C and $\kappa_2 = 1.74^\circ$C.

4.3 Simulation

As the temperature dynamics of the real system are relatively slow with a time-constant of 1075 s, we will use a simulated model that runs ten times as fast during the design phase of the controller. This avoids too much tedious waiting. The animation window, see Figure 2(c), shows the status of the simulated tank. It is also updated when running the real process.

The heat control signal u and the temperature T are plotted in another window, see Figure 2(b). The plots should be used to evaluate the controller performance. The plotting can be halted by clicking `Stop Plot` in `Client Opcom`, see Figure 2(d), and started anew by clicking `Start Plot`.

JGrafchart communicates both with the real and the simulated process. JGrafchart decides whether to simulate or talk to the real process by means of the boolean variable `Simulation` in the `Top` workspace of your JGrafchart controller file, shown in Figure 2(a). If `Simulation` is 1 the simulated model, running ten times as fast as the real process, provides your controller with measurement signals. If `Simulation` is 0 the real world process is used. The necessary time-scaling of parameters is done automatically. (We will see later how to toggle this mode.)

5. Sequential Control

In this part of the laboratory exercise you will develop a sequence for controlling the batch reactor. The sequence should be described as a Grafcet diagram and then translated to JGrafchart. The sequence will be tested against the simulated process and then evaluated against the real process.

Sequential Control of the Batch Reactor

The reactor is making batches over and over again. The making of one batch is outlined below. All buttons and variables live in the `Top` workspace, if nothing else is explicitly stated.

1. The operator starts the batch by pressing the `StartSequence` button, which sets the boolean variable `Start` to 1 while the button is depressed.
2. Once the start button has has been pressed, the reactor should be filled using the in-pump, which is running as long as the boolean variable `InPump` is 1. The filling should be stopped when the boolean variable `Sensor.Full` (from the `Sensor` workspace, which is a sub-workspace of `Top`) becomes 1.

3. As soon as the in-pump is stopped, the agitator should be started by setting `Agitator` to 1 and the heating controller should be turned on by setting `Control_On` to 1.

4. Heating control and agitation should be stopped when the operator presses the button `StopHeating`, which assigns the boolean variable `StopHeat` the value 1. It is the responsibility of the operator to wait until a desired temperature is reached, before pressing the button. When heating and agitation have stopped, the tank should automatically be emptied by means of the out-pump, controlled by the boolean variable `OutPump`. The out-pump should be turned off once `Sensor.Empty` becomes 1.

5. The tank needs to be automatically Cleaned In Place (CIP) before the batch sequence can be repeated. The CIP procedure consists in flushing and cooling the tank: The tank should be filled until the variable `Sensor.Full` becomes 1. Subsequently, the agitator is started and the cooler is activated by setting `Cool` to 1. When temperature has dropped to 25°C, agitations and cooling should stop. Temperature measurements in units of °C are available through the variable `Temp`. Next, the out-pump empties the tank until `Sensor.Empty` becomes 1. Finally, the out-pump is turned off. After this, execution should return to the start state, where the process waits until the operator presses the start button anew.

Note: It might happen in simulation, as well as in the real process, that the temperature is below 25°*C* already at the beginning of the CIP step. If this is the case, the CIP step will only involve flushing of the tank.

Preparation Exercise 5.1 Draw a Grafcet diagram (pen and paper), which describes the sequence above. Use a macro step to implement the CIP.

JGrafchart is different from the Grafcet standard. You need to make some adjustments to the sequence to be able to use it in JGrafchart, cf. Section A.

Preparation Exercise 5.2 Translate your Grafcet sequence in Preparation Exercise 5.1 to the programming language syntax of JGrafchart (still pen and paper). It means that in this exercise you should write (draw) the exact code that is necessary to run the control system. Use the exact names of the variables mentioned above. See Section A for details of the JGrafchart language. Use the skeleton, provided in the file `xml/lab3_start.xml`, shown in the left half of Figure 2(a).

Note: It is recommended that you use the blocks present in the skeleton. However, it is fully possible to add or remove blocks and connections.

Note: Variables in sub-workspaces are accessible by lexical scoping or 'dot'-notation. E.g., since the `Sensor` workspace is a sub-workspace of the `Top` workspace, the `Full` variable in the `Sensor` workspace is accessed by `Sensor.Full` in the `Top` workspace and macro steps defined within it. You can write `Sensor.Full` in the body of the `CIP` macro step of the `Top` workspace to access the `Full` variable of the `Sensor` workspace.

Programming and Simulation of the Control Sequence

Exercise 5.3 *Implement a discrete level sensor in the* `Sensor` *workspace*. You access the `Sensor` workspace by right-clicking on the corresponding rectangle in

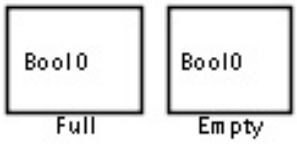


Figure 3 Contents of the `Sensor` workspace template

the `Top` workspace (see Figure 2(a)) and selecting `Show/Hide Body`. The contents of the `Sensor` workspace template, shown in Figure 5, should now be visible in a new window. Your task is to program a JGrafchart sequence, setting the boolean state variables `Full` and `Empty`, using the real variable `Level` (which is defined in the `Top` workspace and available in the `Sensor` workspace by its name, `Level`). The `Full` variable should be set to 1 if and only if `Level >= 0.28` and `Empty` should be 1 if and only if `Level <= 0.025`.

Exercise 5.4 Implement your sequence from Preparation Exercise 5.2. Test it in simulation. Compile the program by selecting `Compile All` from the `Execute` menu (or by using the tool bar). Start the simulation by selecting `Execute` from the same menu (or tool bar). Don't forget to press the `StartSequence` button to start one batch, and the `StopHeating` button to start CIP. (Both are found on the `Top` workspace.)

Note: Execution is stopped by selecting `Stop` from the `Execute` menu.

Note: You have to re-compile and execute each time you make changes.

Note: The boolean variable `Simulation`, found in the `Top` workspace, should be set to 1, which is the default. If you have changed it, press the `Sim_On` button (`Top` workspace) once your program is running.

Evaluation using the Real Process

When the sequence gives a satisfying result in simulation it is time to try it on the real process.

Exercise 5.5 Make sure that the computer and the process are connected and that the LED on the front of the process is green. Set the value of `Simulation` (`Top` workspace) to 0 by clicking the `Sim_Off` button (also `Top` workspace) during execution to use the real process. You might need to calibrate the level sensor, which you have implemented in the `Sensor` workspace. The `Empty` level should correspond to no (or very little) water in the tank and the `Full` level should correspond to water approximately 2 cm above the agitator blades. Make adequate modifications in your `Sensor` workspace for this to happen and evaluate the sequence on the real process.

6. Control of the Temperature

Until now, the temperature has been controlled by means of a Proportional controller (P controller) with proportional gain 1000, making it behave like an on/off controller. We will now investigate how control performance is affected by varying the proportional gain of the P controller. Subsequently, the controller will be extended to a Proportional–Integrating (PI) controller in order to achieve better control performance (in terms of tracking, disturbance rejection and control signal activity).

P control

The `Top` workspace holds the sub-workspace `PID`, containing the macro step `P_Controller`, within which a P controller is implemented. Make sure you understand its implementation prior to proceeding.

The `PID` workspace also contains the `PI_Controller` macro step and logics to direct execution to either the P or PI controller. The `PI_Controller` step will be handled later.

Exercise 6.1 Study how the gain K of the P controller influences the heating behavior. Use $K = 4,\ 15,\ 200$. Set the reference signal `Tref` to 40 °C. Try to explain the observed behavior.

Note: To update the controller parameters it might be necessary to click the `Sim_On` action button on the `Top` after changing the values.

Note: Some of the variables of the `PID` workspace are not used in this lab, since we are not interested in a derivative part. Why are we not interested in derivative action?

PI control

For processes working around a stationary point corresponding to non-zero input signal, P control results in a stationary control error. This error can be decreased by increasing the proportional gain. However, this is done at the expense of stability margins and noise suppression. An attractive alternative to increasing the gain is to introduce an integrator in the controller.

A continuous time PI controller has the transfer function:

$$U(s) = K\left(1 + \frac{1}{sT_i}\right)E(s)$$

where $E(s) = Y_r(s) - Y(s)$. The signals and constants are:

- U control signal
- Y process output
- Y_r reference
- E control error
- K proportional gain
- T_i integral gain

To be able to implement this controller in a computer, the integrating parts must be replaced by a discrete time approximation. Here this is done by assuming constant control error between sample points. Introducing this approximation, one obtains the following pseudo-code implementation (running once per sample period):

```
Ppart = K*(r-y)
v1 = Ppart + Ipart
if v1 <= umin:
  u = umin
else if v1 >= umax:
  u = umax
else:
  u = v1
Ipart = I_Old+K*h/Ti*(r-y)
```

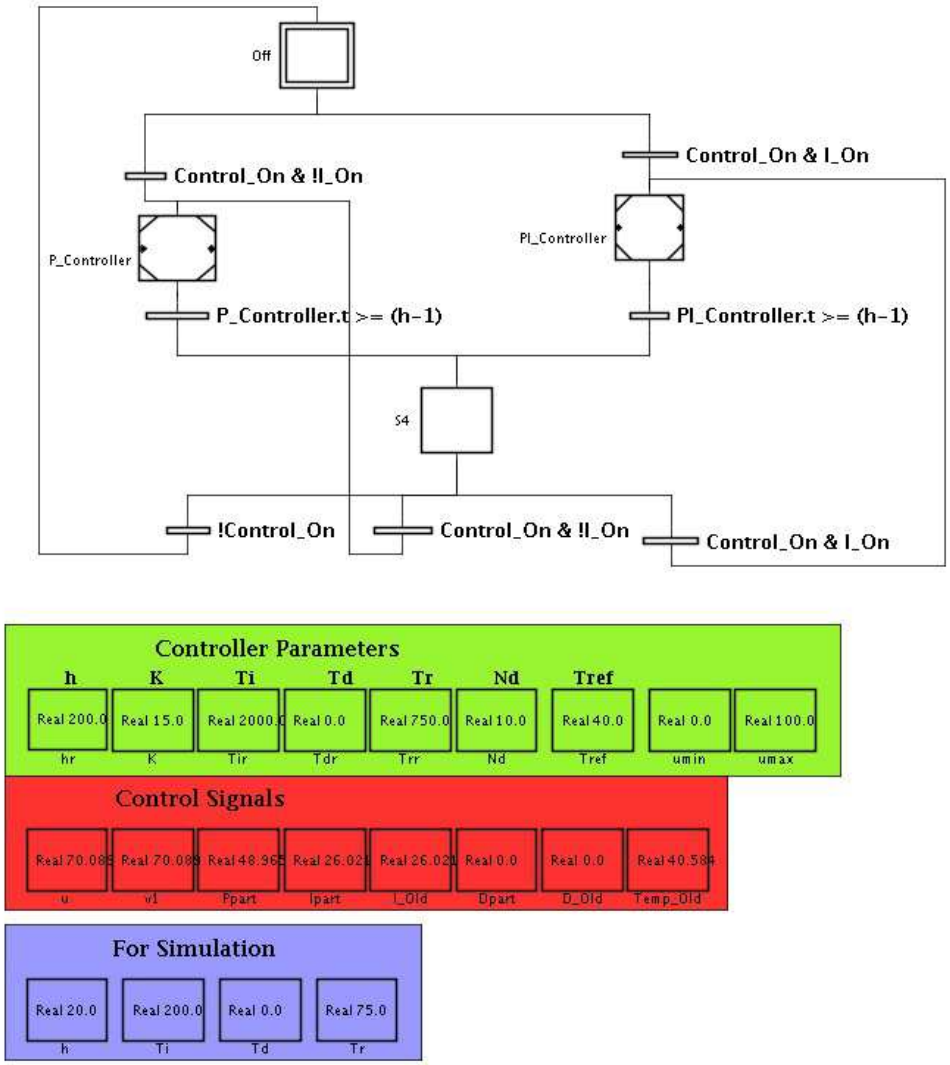


Figure 4 Contents of the `PID` workspace

Preparation Exercise 6.2 Implement the PI algorithm in JGrafchart (pen and paper). Use `Temp` for the latest measurement (`y`) of the temperature. For parameters, use the names `K`, `Ti`, `h`. Calculate the nominal control signal `v1` as the sum of the terms `Ppart` and `Ipart`, which represent the internal values of the controller's proportional and integral parts.

The P controller used until now, provides the blocks and interconnections needed for your PI implementation. Its JGrafchart implementation is shown in Figure 6.

Exercise 6.3 Implement your PI controller from Preparation Exercise 6.2 in the designated macro step in the `PID` workspace (sub-workspace of `Top`) in JGrafchart, see Figure 6.

Note: The PI macro step already contains the P controller shown in Figure 6.

Note: Notice that the temperature controller will only start if the boolean variable `PID_On` (`Top` workspace) is 1 and that the P controller (as opposed to the PI controller) is used if the boolean variable `I_On` (`Top` workspace) is 0.

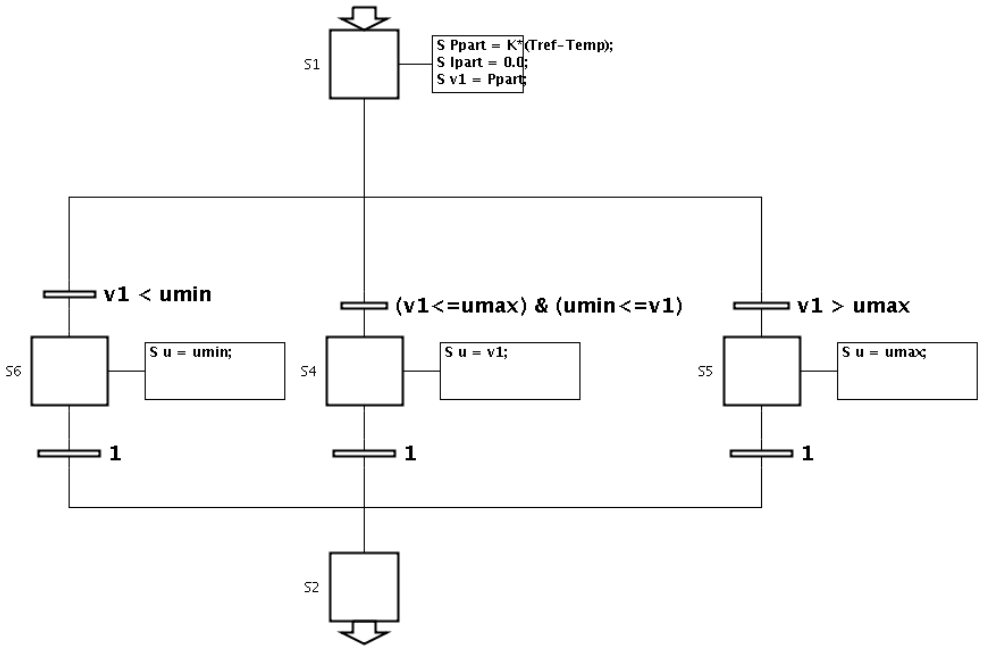


Figure 5 P controller for the `P_Controller` macro step

Test your PI controller

Exercise 6.4 Simulate the system with a PI controller parameterized by $K = 15$, $T_i = 2000$, $T_d = 0$ and $h = 200$. Study the behavior of the system. What happens if the control signal saturates?

In order to avoid the phenomenon mentioned above, a term `h/Tr*(u-v1)`, where `Tr` is a positive *tracking* constant, can be added, when updating `Ipart`. This term prevents `Ipart` from continue growing if the control signal saturates.

Notice that the final control signal `u` is computed from a nominal control signal `v1` in the pseudo-code for the PI controller.

Preparation Exercise 6.5 The phenomenon mentioned above is called integrator windup. Why does it occur and how does it affect the system? Modify your PI controller (pen and paper) to include integral anti-windup.

Exercise 6.6 Change your PI controller sequence in JGrafchart to include the anti-windup. The parameter `Tr` is available on the `Control` workspace. What happens to the control signal when `Tr`=200, 750, and 2500?

7. The Final Result

Exercise 7.1 As a final part of the laboration, run the entire batch sequence, including the windup-protected PI controller, on the real process. Use the controller parameters:

$$K = 15$$
$$Ti = 2000$$
$$h = 200$$
$$Tr = 750$$

(Don't forget to set the variable `Simulation` (`Top` workspace) to 0. Click the `Sim_Off` action button to change this.)

How reliable is the model of the system used for simulation?

8. Conclusions

During the laboratory exercise we have developed a small control program for a batch reactor. The program contains both a sequential part and a PI controller for temperature control. This mix of control loops and logic is very common and can be found in all from highly complex industrial processes to electric domestic appliances such as laundry machines.

A. Introduction to JGrafchart

JGrafchart is a Grafcet/SFC editor and execution environment developed at the Department of Automatic Control, Lund University.

A.1 On-line Help

JGrafchart has an on-line help, which can be reached from the menu choice *Help*. Help about any object in JGrafchart can also be found by pressing the speech bubble with an *"i"* on the toolbar. Once it has been pressed the marker arrow will become a speech bubble with a *"?"*. If the speech bubble with a *"?"* is placed over any object and the mouse button is pressed information about the object will appear in a browser. The browser may take some time to start. To go back to normal mode press the speech bubble with an *"i"* once again. Use the help!

A.2 Workspaces

Grafcet sequence diagrams are created interactively using drag-and-drop from a palette containing the different Grafcet language elements. The sequence diagrams are stored on JGrafchart workspaces, see Figure 6.

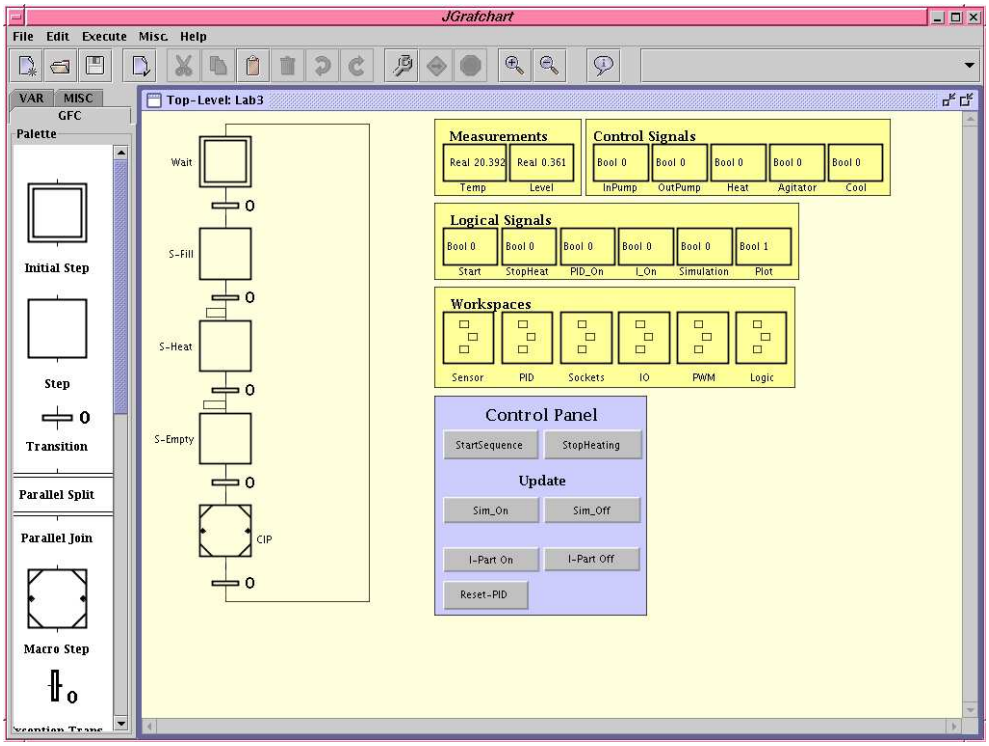


Figure 6 JGrafchart with palette, menus, toolbar, and a workspace.

Workspaces can be stored to a file and loaded from a file. The file is stored using the XML format.

If multiple workspaces are used, only one of them is the current focus for menu choices. This is indicated through a blue workspace border, rather than the ordinary gray border. The focus is changed by clicking on a workspace. This also automatically moves the workspace to the front.

On a workspace it is possible to select an object or an area containing multiple objects in the standard fashion. A selected object can be moved, cut to the clipboard, or copied to the clipboard. The contents of the clipboard can be pasted to a workspace.

Grafcet objects are connected together graphically by clicking on the connection stubs. A connection can be moved by selecting it and moving one of the green corner points. This is specially needed when a Grafcet object is connected to another object that is above the first object.

A selected Grafcet object or connection is deleted using the `Delete` key.

A.3 Grafcet Elements

This version of JGrafchart supports the following Grafchart elements: steps, initial steps, transitions, parallel splits, parallel joins, macro steps, work space objects, exception transitions, digital inputs, digital outputs, analog inputs, analog outputs, socket inputs, socket outputs, internal variables (real, boolean, string, and integer), action buttons, and free text for comments. Inputs and outputs have already been configured for this lab, and are therefore not covered here.

Steps Grafcet steps have action blocks that may be made visible or hidden through menu choices on the step menu that is obtained by double-clicking on the step, see Figure 7. The name of a step is located on the left hand side and it can be changed by click-and-edit. Step actions are entered as text strings, through the *Edit* step menu

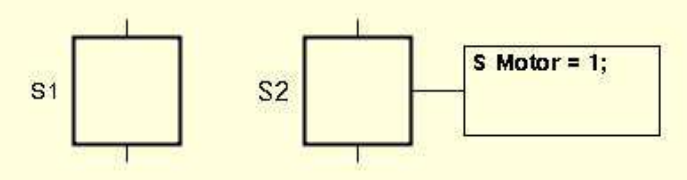


Figure 7 Step with action block hidden and visible.

choice, which is found by right-clicking on the icon of the step. Multiple step actions are separated by semi-colons.

Four different action types are supported. Stored actions (impulse actions) are executed once when the step is activated. The syntax for stored actions is:

```
S "variable-or-output" = "expression";
```

Periodic actions (always actions) are executed periodically, once every scan cycle, while the step is active. The syntax for periodic actions is:

```
P "variable-or-output" = "expression";
```

Exit actions (finally actions) are executed once, immediately before the step is deactivated. The syntax for exit actions is:

```
X "variable-or-output" = "expression";
```

Normal actions (level actions) associate the truth value of a digital output or a boolean variable with the activation status of the step. The syntax for a normal action is:

```
N "output";
```

The expression syntax follows the ordinary Java syntax, with some minor exceptions. One important exception is that the literal 0 is used both to represent the boolean literal False and the integer literal 0. The context decides the interpretation.

The operators supported are: + (plus), - (minus), * (multiplication), / (division), ! (negation), & (and), | (or), == (equal), != (not equal), < (less than), > (greater than), <= (less or equal), >= (greater or equal).

Expressions may contain name references to inputs, outputs, and variables. JGrafchart uses lexical scoping based on workspaces. For example, a variable named X on

workspace W1 is different from a variable named X on workspace W2. References between workspaces are expressed using dot-notation. For example, a step action in a step on workspace W1 can refer to the variable Y on workspace W2 using `W2.Y`.

Initial Steps Initial steps are ordinary steps that are active initially when the execution of the sequence diagram starts. Initial steps may have actions in the same way as ordinary steps.

Transitions Transitions represent conditions or events that should be true (1) in order for the Grafcet to change state. The transition expression is represented by a text string associated with the transition, see Figure 8. A transition expression is edited

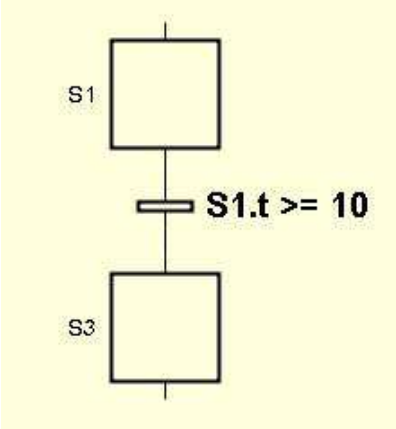


Figure 8 Transition

through the *Edit* transition menu choice (right click on the icon of the transition).

The transition expression should return a boolean value. The expression syntax is the same as for step actions with a few additions. The expression "stepName".x returns 1 if the step is active and 0 otherwise. The expression "stepName".t returns the number of scan cycles since the step last was activated. A scan cycle is typically 10-50 ms. The expression "stepName".s returns the absolute time, in seconds, since the step last was activated. The expression /"boolean-variable-or-input" represents a positive trigger event. It is 1 if the value of the variable or input was 0 in the previous scan cycle and is 1 in the current cycle. Similarly, the expression \"boolean-variable-or-input" represents a negative trigger event. For example, the expression `(/y | \y)` is 1 whenever the boolean variable `y` changes its value.

Parallel Splits and Joins Parallel branches are created and terminated with parallel splits and parallel joins. The parallel objects only allow two parallel branches. If more branches are needed, the parallel elements can be connected in series, see Figure 9.

Macro Steps A macro step represents a hierarchical abstraction. The macro step contains an internal structure of steps, transitions, and macro steps represented on a separate (sub-)workspace. The sub-workspace is made visible and hidden by double-clicking on the macro step. The first step in the macro step is represented by a special enter step. Similarly the final step of the macro step is represented by a special exit step. Both the enter step and exit step are ordinary steps and may, e.g., have actions. The situation is shown in Figure 10.

The sub-workspace of a macro step has a local name space lexically contained within the name space of the macro step itself. For example, the sub-workspace of the macro step M1 may itself contain a macro step named M1, without causing any ambiguities.

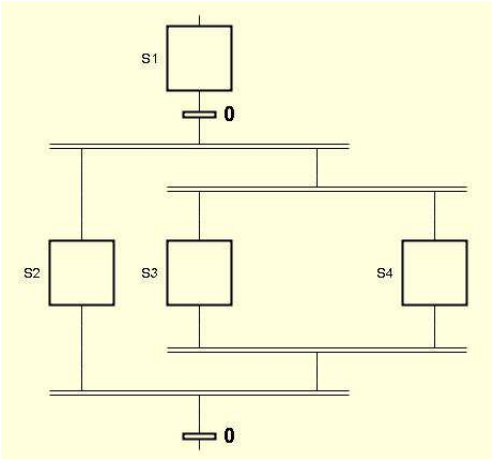


Figure 9 Parallel branching with three branches.

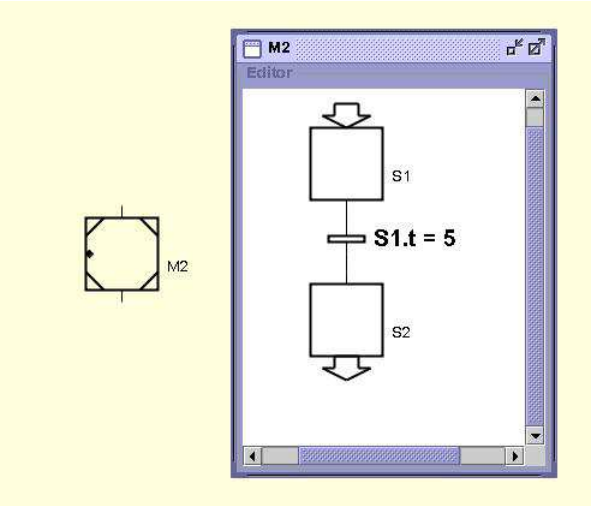


Figure 10 Macro step M2 with internal structure. The internal structure contains the enter step S1 and the exit step S2.

Exception Transitions An exception transition is a special type of transition that only may be connected to a macro step. The exception transition is connected on the left hand side of the macro step. An ordinary transition connected to a macro step does not become enabled until the execution of the macro step has reached the exit step. An exception transition, however, is enabled all the time while the macro step is active. When the transition is fired the execution inside the macro step is terminated and the step succeeding the exception transition becomes activated. Exception transitions have priority over ordinary transitions in cases where both are fire-able. An exception transition connected to a macro step is shown in Figure 11.

Internal Variables Internal variables are variables that can be both read from and written to. Four types of variables are available: real variables, boolean variables, integer variables, and string variables. Associated with each variable are its value and its name, see Figure 12. Both can be changed by click-and-edit.

Action Buttons An action button performs an action when clicked on during execution. The syntax of the action is the same as for stored actions of a step, see Figure 13.

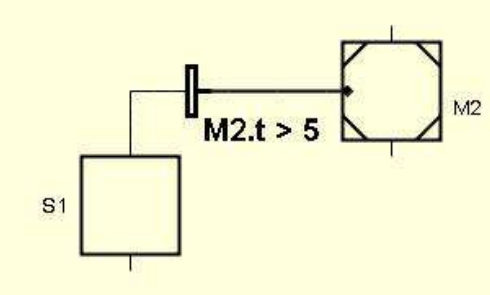


Figure 11 An exception transition connected to macro step M2. The exception transition will fire when M2 has been active longer than 5 scan cycles.

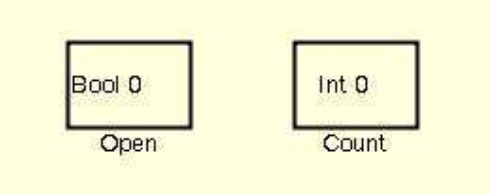


Figure 12 Boolean variable (left) and integer variable (right).

Multiple actions are written on one line separated by semi-colons.

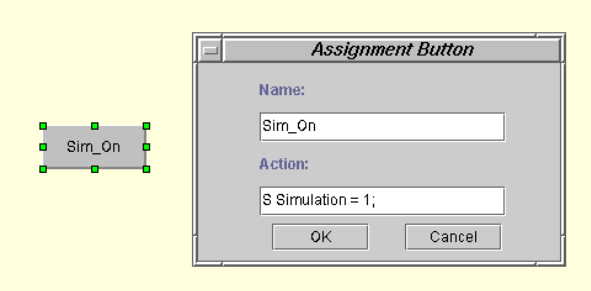


Figure 13 Action Button with its action.

Workspace Object To easier organize the programs in JGrafchart one can use the workspace object. A workspace object on the top level workspace contains a sub-workspace that can be used just as the top level workspace.

Free Text Text comments can be added to a workspace by drag-and-drop of the *Free Text* text string on the palette. By single-clicking on the text the text string can be edited. By double-clicking on the text a menu is shown where it is possible to change font, size, color, etc of the text.

A.4 Execution

Grafcet sequence diagrams are executed by a periodic thread associated with each top-level workspace. The thread cyclically performs three operations:

1. Read Inputs. The values of the digital inputs are read.
2. Execute Diagram. All the transitions in the diagram are checked. Steps are activated and deactivated.

3. Write Outputs. The values of the digital outputs are written.

Before a sequence diagram can be executed it must be compiled. This is done by selecting *Compile All* from the *Execute* menu or by using the buttons on the toolbar, see Figure 6.

Two types of problems may arise during compilation: parsing errors and symbol table lookup errors. Parsing errors are actually detected already when the step actions and transition expressions are entered. For example, the transition expression (y OR z) would generate a parsing error. (The syntactically correct expression should be `(y | z)`). Symbol table lookup errors occur if a name reference does not exist, e.g., if there does not exist any variables named `y` or `z` in the previous example. Both parsing errors and symbol table lookup errors are indicated by a change in the text color of the transition expression or step action from black to red. There will also be an error message written in the field next to the stop button on the toolbar.

In the `Execute Diagram` part of the execution cycle the following operations are performed. For each transition in the diagram, the transition expression is evaluated. If it is 0, then the transition icon is changed to red. If it is 1, the transition icon is changed to green. If, additionally, all steps preceding the transition are active, then the steps preceding the transition is marked to become deactivated in the next cycle, and all the steps succeeding the transition are marked to become activated in the next cycle. When all transitions have been checked, the change of step state is effectuated. In addition to the things above, step actions are executed and the step timing information is updated.

Programming in JGrafchart

Programming in JGrafchart is done by dragging and dropping object from the palette in Figure 6 on to a workspace. The objects are connected by clicking on the stubs of and object and drawing a line to the object it is to be connected with. The rules of Grafcet has to be followed: A step cannot be connected to a step and so on. The steps and transitions can be edited. Variables and inputs and outputs are defined by dragging and dropping them on a workspace. After the function diagram has been built the workspace has to be compiled and if there are no errors the function chart can be executed.