Market Driven Systems

Laboratory Exercise 1

Implementation of a Batch Process Control System

Department of Automatic Control
Lund Institute of Technology
Last update August 2011

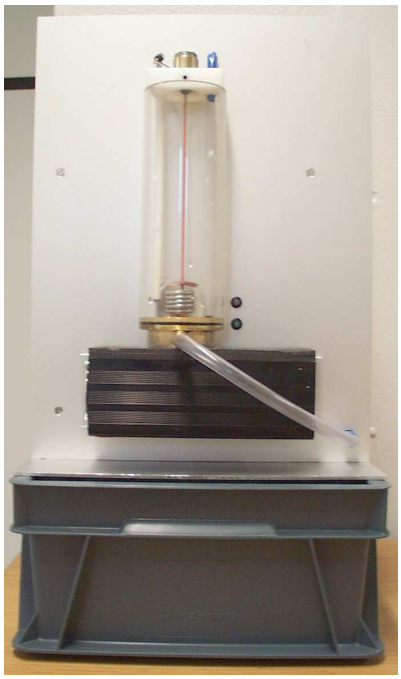


Figure 1 The batch reactor.

1. 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.

2. Getting started

Login as `lab_batch`, no password.

Open a terminal window and start JGrafchart with the command `JGrafchart`.

Rather than starting from scratch, open `start.xml` in JGrafchart, it contains some definitions and structure. You should now have the same windows as in figure 2.

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.

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 application 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 and one for emptying the tank. They are controlled using the boolean variables `InPump` and `OutPump` respectively.

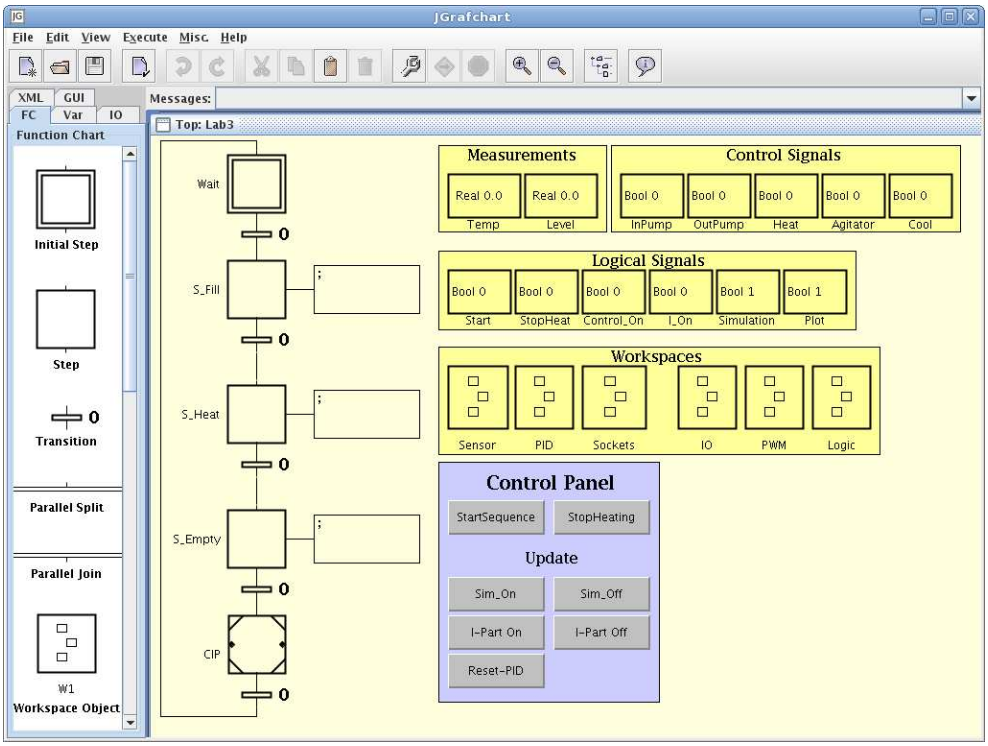
It is also possible to start or stop the pump, cooler and agitator from the `Client OpCom` window shown in figure 2(d).

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. A larger number corresponds to higher water level. `Temp` is in the range 0–100, which corresponds to 0–100 °C.

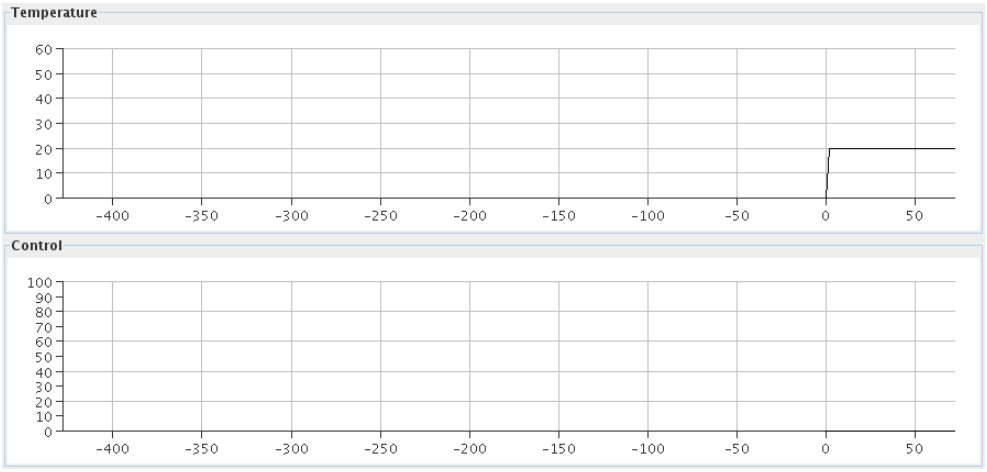
The process has some interlocks built in, e.g. it is not possible to heat the tank if the level is too low. In the case something prohibited takes place the process sets the `Error` signal variable to 1 and the LED on the front of the process turns red. If this happens, try to resolve the cause and then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

4. Modeling and Simulation

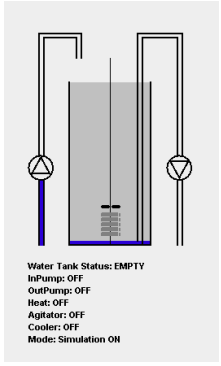
We will control two states in the batch reactor: the water level and the temperature of the water.



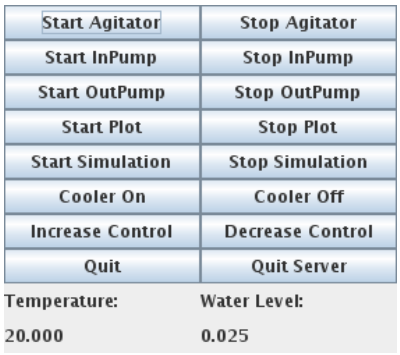
(a) JGrafchart with the Top workspace of the application showing.



(b) Plotter



(c) Tank animation



(d) Client OpCom

Figure 2 Windows of the graphical user interface.

4.1 Water Level Dynamics

Let us denote the water level as a function of time with $h(t)$ in meters. Mass conservation gives the differential equation

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

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 signal 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 the JGrafchart application, 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.
   Note: `Start` is automatically reset by the `Logic` workspace.
2. Once the start button 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 provided skeleton, 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 with the dot-notation, e.g. the `Full` variable in the `Sensor` workspace is accessed with `Sensor.Full`. You can write `Sensor.Full` in the body of the `CIP` macro step too since JGrafchart uses lexical scoping and automatically looks in the enclosing workspace if there is no match locally.

Programming and Simulation of the Control Sequence

Exercise 5.3 *Implement a discrete level sensor in the* `Sensor` *workspace.* A workspace is opened by right-clicking on it and selecting `Show/Hide Body`. The contents of the `Sensor` workspace template, shown in figure 3, should now be visible in a new window. Your task is to program a JGrafchart sequence, setting the boolean 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` from the `Execute` menu (same as the monkey wrench in the toolbar). Start the simulation by selecting `Execute` from the same menu (or the sign with a right-arrow in the toolbar). Don't forget

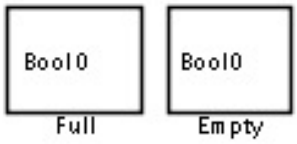


Figure 3 Contents of the `Sensor` workspace template

to press the `StartSequence` button to start one batch, and the `StopHeating` button to start CIP.

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

Note: You have to stop, 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 when 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 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.

Note: To see the real process values in the Plotter you also need to click `Stop Simulation` in `Client OpCom`.

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`, in 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 Turn simulation back on. 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: 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)
```

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 5.

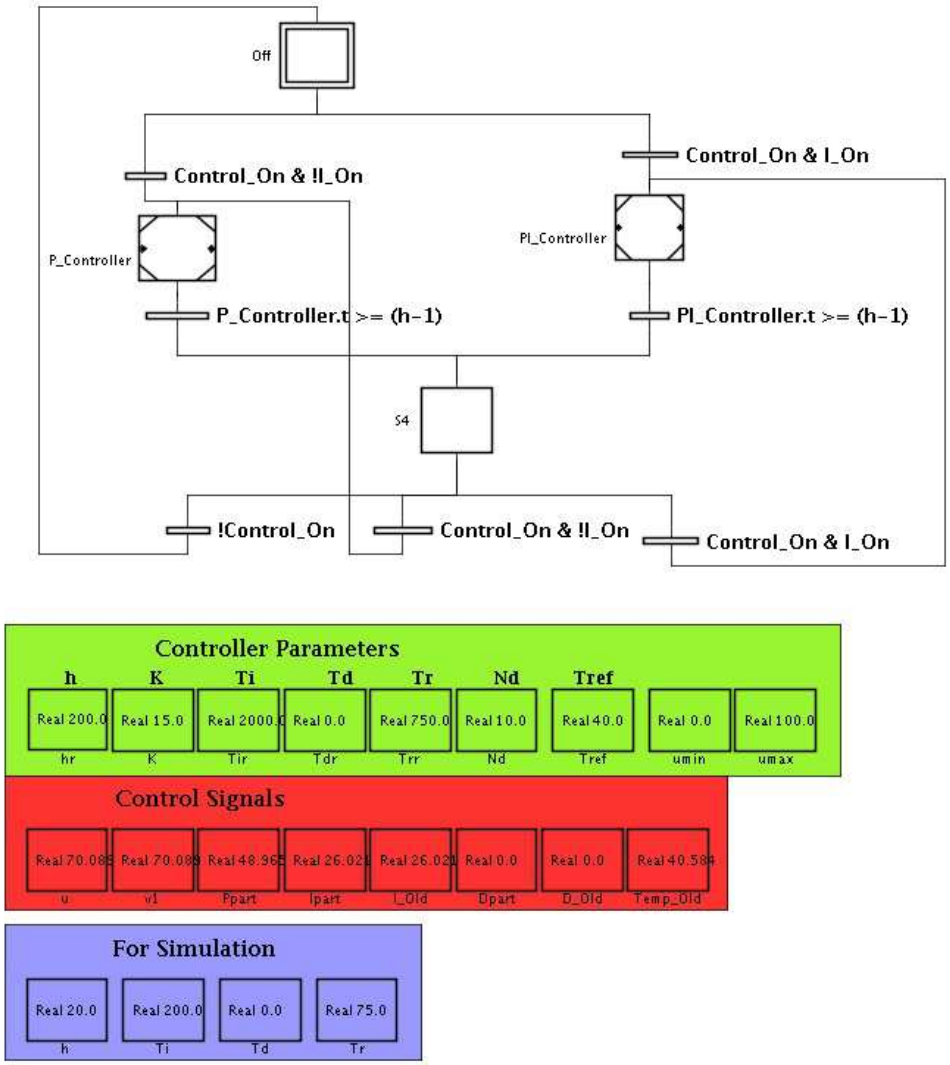


Figure 4 Contents of the `PID` workspace

Exercise 6.3 Implement your PI controller from Preparation Exercise 6.2 in the designated macro step in the `PID` workspace, see figure 4.

Note: The PI macro step already contains the P controller shown in figure 5.

Note: Notice that the temperature controller will only start if the boolean variable `Control_On` is 1, and that the P controller (as opposed to the PI controller) is used if the boolean variable `I_On` is 0. To manipulate `I_On`, use the `I-Part On` and `I-Part Off` buttons.

Test your PI controller

Exercise 6.4 Simulate the system with a PI controller parametrized by $K = 15$, $T_i = 2000$, $T_d = 0$ and $h = 20$. 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.

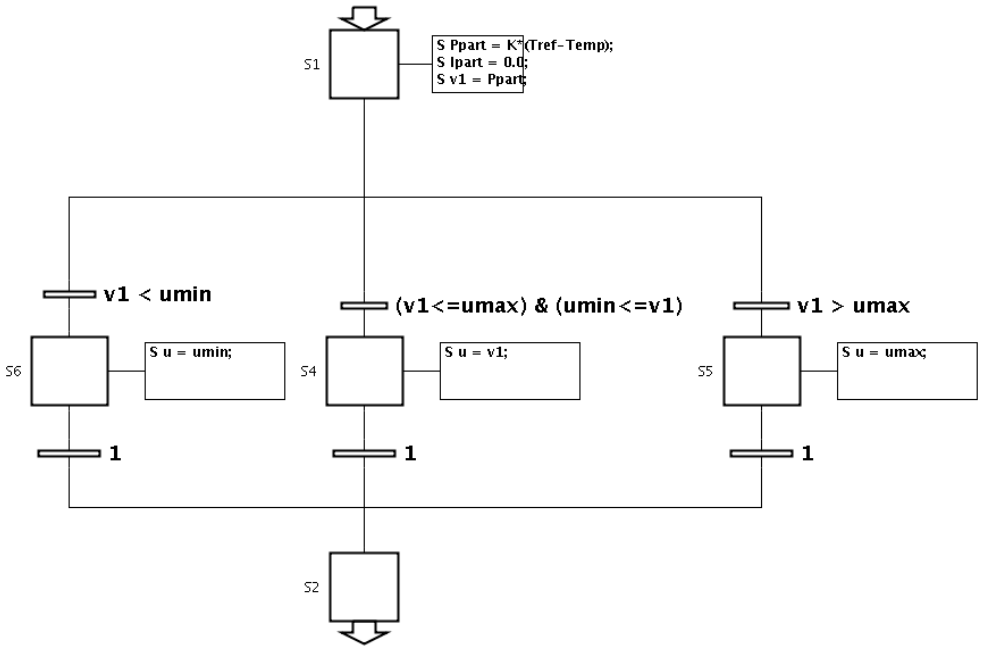


Figure 5 P controller for the `P_Controller` macro step

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 lab, run the entire batch sequence, including the windup-protected PI controller, on the real process. Use the controller parameters:

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

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 freeware function chart editor and execution environment developed at the Department of Automatic Control, Lund University.

In this lab, only a subset of all elements in JGrafchart are available, and only what you need to know to be able to do the lab is described in this section.

A.1 Programming in JGrafchart

Programming in JGrafchart is done by drag-and-drop from the palette to a workspace. The palette contains steps, transitions, variables, and lots of other elements. It is possible to select, delete, copy, cut, and paste objects in the standard fashion. Function chart objects are connected by click-dragging the stubs. The rules of Grafcet have to be followed, e.g. a step cannot be connected to a step.

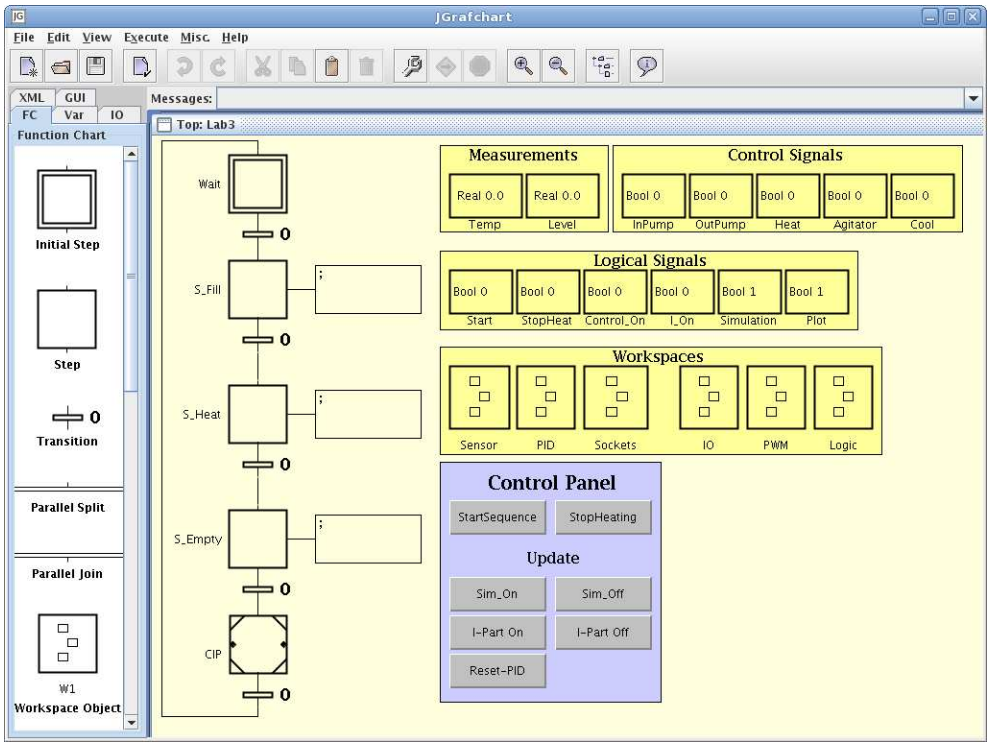


Figure 6 JGrafchart with palette (left) and the Top workspace of an application (right).

A.2 Documentation

For more detailed information, use the documentation for JGrafchart which can be opened with *Online Help* in the *Help* menu. The documentation for an object can also be consulted interactively by using the *Object Help* feature on it, i.e. the speech bubble with an *i* in the toolbar or *Object Help* in the *Help* menu.

A.3 Execution

JGrafchart applications are executed periodically. Every period (scan-cycle) three operations are performed:

1. Read digital and analog inputs.
2. Execute one scan of the application.

3. Update variables subject to normal actions.

Before an application can be executed it must be compiled. This is done by selecting *Compile* in the *Execute* menu or in the toolbar. Compilation errors are indicated by red text color of the condition expression/step actions. Error messages are also written to the message list.

Two types of problems may arise during compilation: syntactic errors and semantic errors.

For example, the condition expression `y OR z` would generate a syntactic error since two consecutive variables, i.e. `y` and `OR`, are not allowed. What the programmer probably meant here was `y | z`.

Semantic errors means that the syntax is correct but what you are trying to do is not possible. One example is trying to assign to an input. Another example is if name lookup fails, e.g. if there is not variable named `y` or `z` in the previous example.

A.4 Grafcet Elements

In this section a selection of Grafchart elements are described.

Steps The name of a step is located on the left hand side and can be changed by click-and-edit.

Step actions are entered as text in the dialog that is opened with *Edit* in the step's context menu. Actions are separated by semi-colons.

Four different action types are supported:

- **Enter action** (Stored action): The action is executed once when the step becomes active.
  `S "action";`
- **Periodic action**: The action is executed periodically, once every scan-cycle, while the step is active.
  `P "action";`
- **Exit action**: The action is executed once immediately before the step is deactivated.
  `X "action";`
- **Normal action** (Level action): A normal action is used to associate the truth-value of a boolean variable with the activation status of the step.
  `N "boolean variable";`

The expression syntax is similar to Java. One important difference is that the literals 0 and 1 are used both for the boolean values true and false and for the integer values 0 and 1. It is the context that decides the interpretation.

Supported operators are: +, -, *, /, ! (negation), & (and), | (or), == (equal), != (not equal), <, >, <=, >=.

Expressions may contain name references to variables. JGrafchart uses lexical scoping based on workspaces. For example, a variable named Y on workspace W1 is different from a variable named Y 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 with `W2.Y`.

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 was last activated, or 0 if it is not active. The expression "stepName".s returns the time, in seconds, since the step was last activated, or 0 if it is not active.

Initial Steps Initial steps are steps that get actived when the execution of the function chart starts.

Transitions Transitions are associated with conditions or events that should be true in order for the function chart to change state, see figure 7. Click on the condition to edit it.

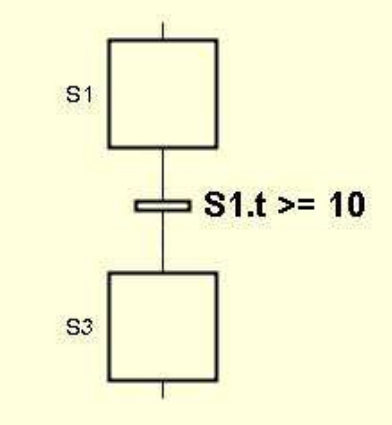


Figure 7 Transition

The condition expression has the same syntax as expressions in step actions and should return a boolean value.

Macro Steps A macro step represents a hierarchical abstraction and contains its own (sub-)workspace, see figure 8. The sub-workspace is opened/closed with `Show/Hide Body` in the context menu. 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 otherwise ordinary steps and may, e.g., have actions.

Variables There is a variable type for each of the primitive data types: real, boolean, integer, and string. Each variable has a value and a name, both can be changed by click-and-edit.

Action Buttons An action button performs actions when clicked during execution, see figure 9. The syntax is the same as for steps, except that only S actions are allowed.

Workspace Object A Workspace Object contains a subworkspace and is typically used to structure the application.

Text *Text* objects are commonly used as comments. The text is changed with click-and-edit.

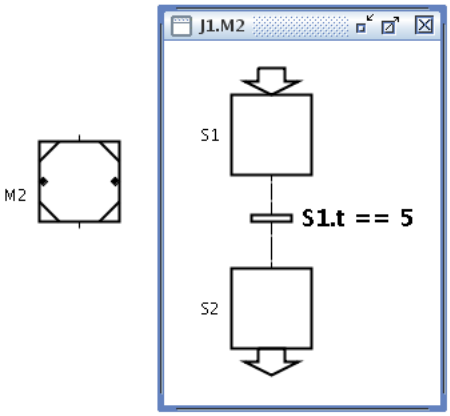


Figure 8 Macro step M2 and its subworkspace containing the enter step S1, the exit step S2 and a transition.

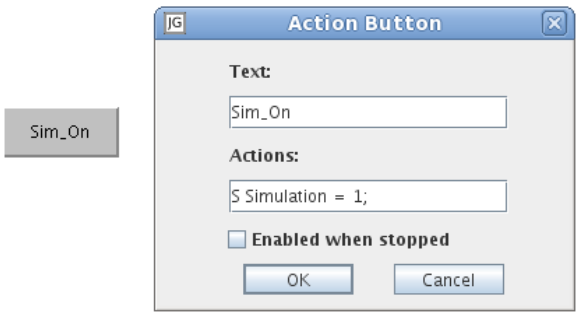


Figure 9 An Action Button with its action.