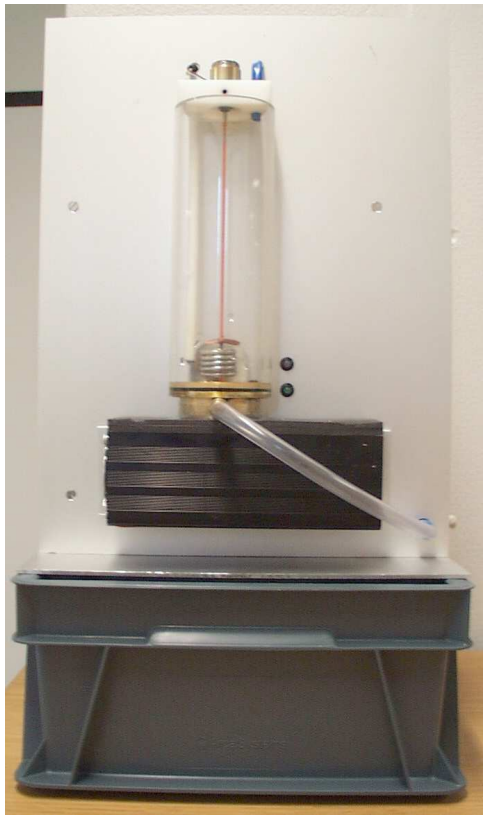
Mini-Project in FRTN25 Process Control

Control of a CSTR Process

Olof Garpinger, Martin Hast, and Ola Johnsson
Department of Automatic Control
Lund University
Latest update: April 2015



1. Introduction

This project concerns control of a continuous stirred tank reactor (CSTR). The project encompasses modeling, controller implementation, controller tuning, and sequence control of a system with multiple inputs and outputs (MIMO).

The process is a tank with an inflow and an outflow, where the inflow is assumed to contain a nutrient medium that should be heated to 37 °C before the next process step; in the actual laboratory process water will be used. The temperature can be manipulated by a heater, and to ensure a homogeneous temperature of the medium, an an agitator can be used. The outflow from the tank is assumed to vary depending on the demand from downstream process steps. The demand will be viewed as a disturbance, and the control must be able to handle this disturbance so that both the temperature and the liquid level in the tank are affected as little as possible.

The tuning of the regulator parameters for the temperature and liquid level controls will be performed using methods that are well established within the

process industry. These methods are based on simple process models that you will derive experimentally.

To automate the start-up, operation, and finalization of the process, you will use sequence control. Based on a given template, you will create a program for control of the CSTR process. The control sequence can be divided into three steps:

1. Initialization step
2. Main step
3. Termination step

In the initialization step, the process should be prepared for continuous operation in the main step. This involves bringing the process state to the desired operating point. In the main step, a given disturbance sequence will affect the system. Using feedback control, the system should be able to handle and suppress the disturbances. Finally, in the termination step, the controls should be switched off, the tank should be emptied and then cleaned in place.

Project Goals

The first goal of the project is that you gain practical experience from sequence control and digital control of MIMO systems. The second goal is that you should learn how to use simple modeling tools and controller tuning methods that are common practice in the process industry.

Project Report and Oral Presentation

When all tasks in this manual have been completed, the final program should be demonstrated to and approved by your project supervisor. The project report should then be completed and handed in, together with the final program. You can use the same format for the project report as for the previous hand-in assignments in the course or write it in the form of a laboratory report. The number of the laboratory process that you have used should be specified in the report. The report should contain all the relevant experimental data, preferably in the form of graphs, that are needed to motivate your statements.

The project will also be presented orally in discussion groups, where the work performed and the results should be discussed.

Preparations

Before you start working, you should read the **complete** project manual, including the section that describes how JGrafchart works, see Appendix A.

2. Software

The development of the control system for the CSTR process will be done in JGrafchart, which is a graphical Java implementation of the GRAFCET language. The starting point for your work is the file `start.xml`, which contains the basic structure of the program that you will develop during the course of the project. For logging of data from the process, you will use the separate program `SockLog`. These files can be downloaded from the course homepage. Below we describe the workflow for various procedures in the form of enumerated lists.

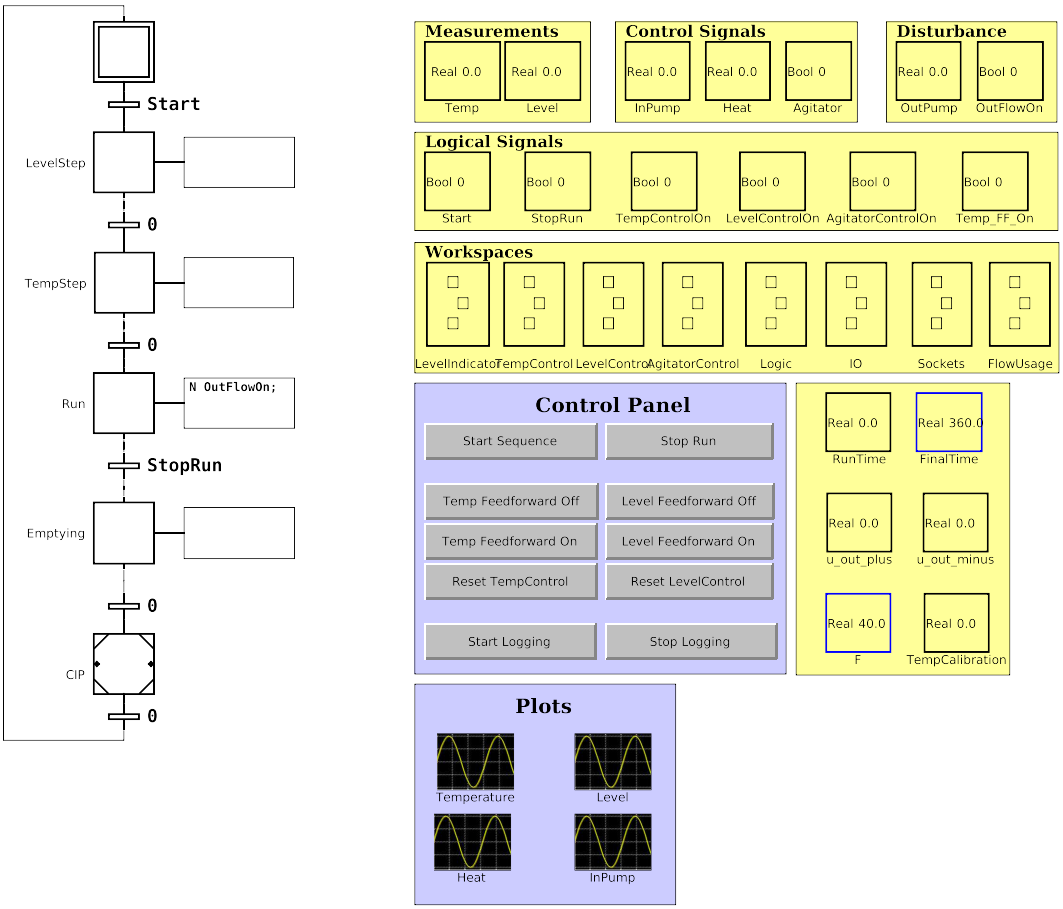


Figure 1 The main workspace in start.xml.

Starting Your Project Work

1. Log onto a computer in the laboratory using your student account.
2. Start a web browser and download the files `start.xml`, `SockLog.sh`, and `SockLog.jar` from the course homepage.
3. Start JGrafchart by opening a terminal window (e.g., Konsole) and typing `JGrafchart`.
4. Open `start.xml` in JGrafchart.
5. Verify that the interface looks like the one shown in Figure 1.
6. Click on the *Properties* button and verify that *Simulator Mode* is unmarked, see Figure 2.
7. Click on the *OK* button to return to the main window.

Executing Your Program

1. Edit your program.
2. Press the *Compile* button to create a new executable. Make sure that no compilation errors have occurred.
3. Press the *Execute* button to start the execution of the program. (An error message will appear if the SockLog program has not been started first, see below. If you do not intend to use logging, you can click *Ignore* to continue.)

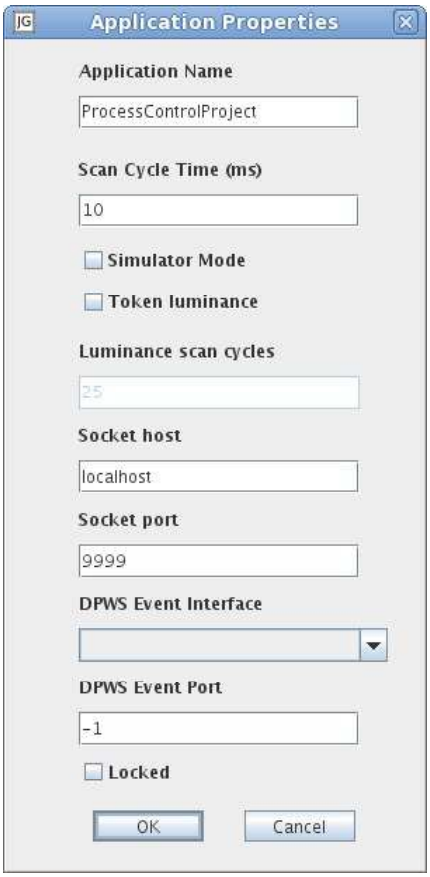


Figure 2 The *Application Properties* window.

4. Perform your experiments.
5. Press the *Stop* button to stop the execution.

Logging and Analyzing Data

1. Open a new terminal window and change directory to where SockLog has been saved.
2. Run SockLog using the command `sh SockLog.sh`.
3. Execute your program according to the instructions above.
4. Press the *Start logging* button.
5. Perform your experiments.
6. Press the *Stop logging* button, give a filename and press save.
7. Repeat from 4 if you want to perform more experiments.
8. Open and run the saved files in MATLAB and analyze your data.

Please note that only the variables that have changed value during logging will be saved. The number of variables in the resulting file may hence vary.

3. The CSTR Process

All experiments will be performed on the laboratory processes at the Department of Automatic Control. There are six processes, and each group is recommended

to stick to the same laboratory process throughout the project, since there are small differences in the dynamics of the various processes.

Before use, the process must be connected to the computer and to an electrical outlet. The process is turned on using the switch on its side. Check the liquid level in the water basin; it should cover the larger part of the inlet pump. An LED indicator on the front of the process shows green when the process is ready and red when an error has occurred. The most common errors result from trying to send a control signal that violates any of the internal interlocks:

- The liquid level in the tank must be above the coil when the heater is turned on, in order not to damage the heater.
- No more liquid may be added if the tank is almost full, in order to prevent flooding.

There is a reset button on the side of the process that can be pressed when the cause of the error has been removed.

Do not forget to turn off the power to the process when you leave the laboratory.

Signals

The process consists of a tank with a number of sensors and actuators. These are illustrated in Figure 3. The measurement signals are:

1. The liquid temperature, `Temp`.
2. The liquid level, `Level`.

The input signals are the voltages over the inlet pump, the outlet pump, the agitator, and the heater. Since we in the given scenario cannot affect the outflow, it will be viewed as a disturbance acting on the process. The following three signals will be available to the controller:

1. The voltage over the inlet pump, `InPump`.
2. The voltage over the heater, `Heat`.
3. The voltage over the agitator, `Agitator`.

The agitator will either be turned on using a predefined voltage or turned off.

In the process industry, signal levels are often not represented by their physical quantities but are expressed in terms of a percentage (0–100%). In the case of the CSTR process, a liquid level at 0% corresponds to an empty tank, while 100% means a full tank. Temperature is measured in a similar way, but here, a value of 0% corresponds to 0 °C, while 100% means 100 °C.[1]

The temperature sensor in the tank has a bias that must be calibrated for.

TASK 1
Calibrate the temperature sensor by changing the value of `TempCalibration` so that `Temp` shows about 20 °C. □

[1] Since the heater has such a low power, the `Temp` signal reports a higher value than the actual value. In this way, a higher power output than the real one is simulated.

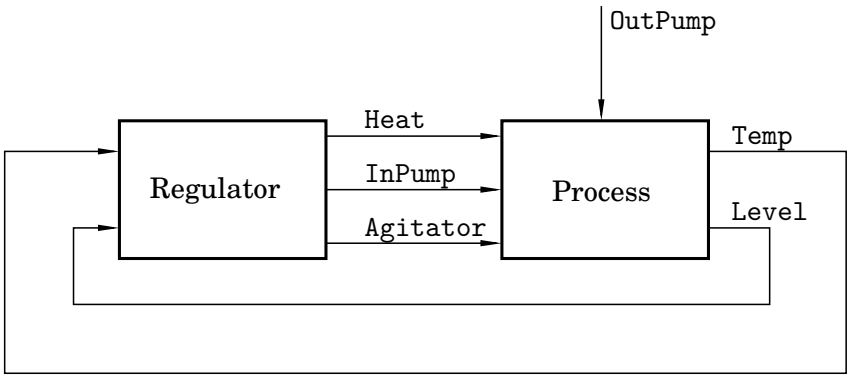


Figure 3 Block diagram over the system and its signals. The names of the signals are the same as those used in the JGrafchart template.

TASK 2
Reason about how the control signals affect the measurement signals. Discuss and motivate which input signal should be used to control the liquid level and which should be used to control the temperature. □

4. Modeling of the Liquid Level Dynamics

The rate of change of the liquid level, $\mathrm{d}V/\mathrm{d}t$, equals the difference between the inflow q_{in} and the outflow q_{out}, which is described by the mass balance equation

$$\frac{\mathrm{d}V}{\mathrm{d}t} = q_{\text{in}} - q_{\text{out}} \quad [\mathrm{m}^3/\mathrm{s}] \tag{1}$$

Assume that the inflow and the outflow are proportional to the voltages sent to the pumps, i.e., $q_{\text{in}} = k_1 u_{\text{in}}$ and $q_{\text{out}} = k_2 u_{\text{out}}$.

TASK 3
Derive the transfer functions from both pumps to the liquid level in the tank, h. Give expressions for the gain from the inlet pump to the change of level (κ_1) and from the outlet pump to the change of level (κ_2). □

Both pumps have a threshold voltage, i.e., a signal level under which the pump will not transport any liquid. Because of the design of the pump, the threshold voltage could vary depending on whether liquid is already flowing through the pump or not.

The threshold voltage can be determined from a simple experiment. Set the input signal to the pump so that it produces a flow. Then gradually decrease the input signal until the flow stops. The voltage value when the flow has just stopped corresponds to the threshold voltage. The experiment could be repeated with smaller decrements to achieve higher accuracy.

TASK 4
Perform experiments on the inlet and outlet pumps and determine the threshold voltages of both pumps. To do this, execute the program and set the output values (0–100%) by changing `InPump` and `OutPump`, respectively.

Later in the project, in the main step, the outlet pump should operate in a region between 5 and 10 percentage points above the threshold voltage. These voltage

levels are denoted u_{out}^- and u_{out}^+, respectively, i.e.

$$\begin{aligned} \texttt{u\_out\_minus} &= \texttt{u\_out\_threshold + 5} \\ \texttt{u\_out\_plus} &= \texttt{u\_out\_threshold + 10} \end{aligned} \tag{2}$$

Specify values for these parameters in the main window of the program template. □

The pump dynamics are nonlinear, not least because of the threshold voltage discussed above. In the limited interval that the pump will operate, however, the linear model developed in Task 3 can be used. Therefore it is important that the model parameters are determined for the actual operating point that will be used.

If the pumps have been idle for a long period of time or if you by accident have let air into the pumps, then the initial flow will be lower than during continuous operation.

A simple method to determine the gain is to start from the differential equation that describes the liquid height in the tank, see Task 3. Integrating both sides of the equation from time t_0 to time t_1 yields

$$h(t_1) - h(t_0) = \kappa_1 \int_{t_0}^{t_1} u_{\text{in}}(t)\mathrm{d}t - \kappa_2 \int_{t_0}^{t_1} u_{\text{out}}(t)\mathrm{d}t. \tag{3}$$

Task 5
Propose an experiment that avoids the nonlinear dynamics of the pumps and uses (3) to determine κ_1 and κ_2. □

Task 6
Select suitable values for the input signals u_{in} and u_{out}, perform the experiment you proposed above, and determine κ_1 and κ_2. Verify by further experiments that the parameters are valid also for other input signals close to the operating points. Use the possibilities to log and analyze data in MATLAB and to generate graphs for your report. □

5. Liquid Level Control

From Equation (1) we can see that the level dynamics is an integrator. A simple method for design of PI controllers for integrating processes is *arrest time tuning*, see Appendix B.

Task 7
Use arrest time tuning to determine suitable parameters for the level control. Use the arrest time $T_a = 2$ s. □

Task 8
What limits how low arrest time T_a can be specified? Reason about this. □

When implementing the controller, the knowledge of the regulator parameters is not enough. We also need to limit the control signal in order not to damage the

actuator. Also, if the controller has integral action, we need to avoid integrator windup.

In `start.xml`, P control of the liquid level is implemented in the workspace named `LevelControl`. In the macro step `PI` its structure can be observed. In a first step, the desired control signal, $v1$, is calculated. If this signal is below the lower limit or above the upper limit of the actuator, the signal is then saturated.

TASK 9
Implement PI control of the liquid level in JGrafchart. Use a setpoint value of 40%. The expression for updating the I-part, when using the Euler forward discretization method, becomes

```
Ipart = Ipart + K*h/Ti*(Lref-Level) + h/Tt*(u-v1)
```

Note that the expression contains an anti-windup mechanism. Use the rule of thumb $T_t = 0.5T_i$ to avoid having to tune another controller parameter. The control signal should be limited to the interval [`umin`, 100%].

To implement the level control:

1. Open `LevelControl`.
2. Set `umin` to a value just below the threshold voltage of the inlet pump. It is important that the inlet pump does not transport any water when the control signal is `umin`.
3. Specify values for the parameters K, T_i, T_t, and the setpoint L_{ref}.
4. Open the module `PI` inside `LevelControl`.
5. Modify the contents of `PI` to give PI control.

When the level control has been implemented, it can be activated by setting the boolean variable `LevelControlOn` to 1. To turn off the control, reset this variable back to 0. If necessary, adjust T_a (and thereby the controller parameters) until a step change in the outlet pump from u_{out}^- to u_{out}^+ does not give a larger level deviation than 2% of the height of the tank. Remember that T_a is just a design parameter than has been used to derive controller parameters from a model; the real arrest time may be different from the theoretical value if there are model errors. How well does theory match reality regarding the value of T_a? □

6. Modeling of the Temperature Dynamics

One of the great benefits of feedback control is that we can often achieve good performance even if we base the controller design on a rough process model. In the case of liquid level control, we could easily derive a mathematical model based on first principles. Here will instead obtain a simple process model through a step-response experiment.

A common model for many systems in the process industry is a first-order linear model with a deadtime:

$$G_p(s) = \frac{K_p}{sT + 1} e^{-sL} \tag{4}$$

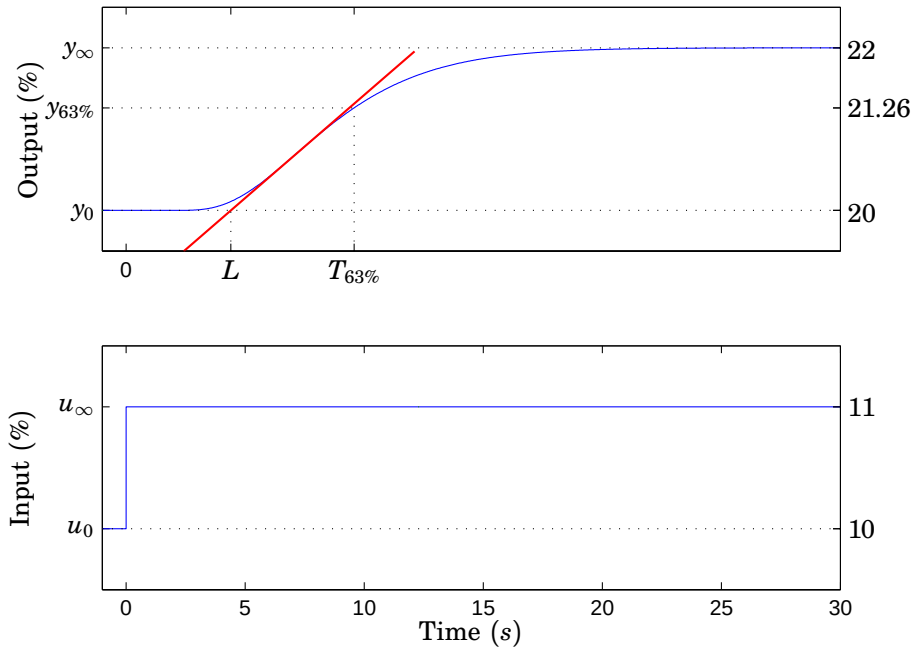


Figure 4 Step response and modeling of a first-order process with a deadtime.

To determine the parameters of the model, a step-response experiment can be carried out. First, the process output should be stabilized near the desired operating point. When all signals are stationary, a step is added to the input signal. The idea is illustrated in Figure 4, where the desired operating point for the output signal is 20%. A step of size 1% has been added to the input signal at time zero, yielding the response shown in the upper plot.

To find the three process parameters in (4), the following procedure can be applied:

1. Determine the process static gain through
$$K_p = \frac{\Delta y}{\Delta u} = \frac{y_\infty - y_0}{u_\infty - u_0} \tag{5}$$
2. Draw a tangent to the output where the slope reaches its maximum value, corresponding to the red line in Figure 4.
3. Determine the deadtime L as the time from the change in the input to the intersection of the tangent to the output base level, y_0.
4. Determine the time, $T_{63\%}$, at which y has reached 63% of the full response, i.e., $y(T_{63\%}) = y_0 + 0.63\,(y_\infty - y_0)$.
5. Determine the time constant T by subtracting the dead time from the time that was measured in the previous step, $T = T_{63\%} - L$.

For the example in Figure 4 we obtain the following values:

$$\begin{aligned} K_p &= \frac{2\%}{1\%} = 2 \\ L &= 4.40 - 0 = 4.40 \\ T &= 9.58 - 4.4 = 5.18 \end{aligned}$$

The process can hence be modeled by the transfer function

$$G_p(s) = \frac{2}{5.18s + 1} e^{-4.4s}. \tag{6}$$

TASK 10
Perform a step-response experiment according to the description above and calculate the model parameters in (4). The input to the heater is given by the variable `Heat` (0–100%). Make sure that there is a continuous flow through the reactor. This can be done by setting the outlet pump value to `u_out_minus` and activating the level control. To achieve a homogeneous temperature in the liquid, the agitator must be turned on. This is done by setting the variable `AgitatorControlOn` to 1.

The experiment should be performed around the desired operating point, which is 37 °C. One way to do this as follows:

1. Find a value of `Heat` that gives the temperature 37 °C in steady state and denote this value u_0.
2. Set `Heat` to $0.8u_0$ and let the temperature stabilize.
3. Make a step change in `Heat` up to $1.2u_0$ and let the temperature stabilize. This gives a step response around 37 °C that can be used in the modeling.
4. If necessary, repeat steps 2–4 with larger steps.

In the report you should clearly show how K_p, L, and T have determined. Use the possibilities to log data and then analyze them in MATLAB. □

7. Temperature Control

A common method for PI controller tuning in process industry is lambda tuning. It assumes that the process can be approximated by a first-order model with a deadtime, see (4). Lambda tuning is described in Lecture 9 and in Chapter 8 of *Process Control*.

TASK 11
Use lambda tuning to derive suitable controller parameters based on your model. Use $\lambda = T$ as a starting point for the desired time constant for the closed-loop system. □

TASK 12
Implement the temperature control in JGrafchart with the setpoint value 37%. There should be a continuous flow through the tank and the agitator should be activated, as in Task 10. If necessary, adjust λ (and hence the controller parameters) if the control gives rise to oscillations or if the closed-loop response is deemed too slow.

To implement the controller, proceed similarly to Task 9, but use the workspace named `TempControl`. □

TASK 13
Make a step disturbance using the outlet pump, changing its value from u_{out}^{-} to u_{out}^{+}. How large is the maximum error in the temperature? Propose at least two different actions that could be taken to reduce the influence of the disturbance. □

8. Feedforward

If it is possible to measure the signal acting as process disturbance, then the liquid level control can be improved by feedforward. The simplest variant of this is a so called static feedforward compensator, which means adding a term to the control signal that is proportional to the measured disturbance.

TASK 14
Starting from the transfer function derived in Task 3, design a static feedforward from the outlet pump signal (the measured disturbance) to the inlet pump signal (the control signal), such that the influence of the disturbance is eliminated. How is the magnitude of the feedforward related to κ_1 and κ_2? How should the feedforward be added to the control algorithm, taking into account control signal saturation and anti-windup? □

The static feedforward can be implemented inside `LevelControl` by specifying a value for the feedforward gain `FF`. Copy the PI controller for level control to the module `PIandFF` and add the feedforward term there. In this way, it will be easy to turn on and off the feedforward using the buttons **Level Feedforward On/Off** in the main workspace.

TASK 15
Implement the feedforward and run the process with the feedforward active and using an outflow of u_{out}^{-}. Make a step-change in the outflow with the amplitude of 20 percentage units. How large is the maximum control error, and how does it compare to the case without feedforward. Make use of the possibilities to log data and then analyze it in MATLAB to generate graphs for your report.

What happens when the process is first run without feedforward and then the feedforward is turned on? Why does this happen and how could it be avoided? □

9. Completing the Control Sequence

All regulator parameters that should be used to control the process are now known, and it is time to merge everything into a complete program. This should be done step by step in the JGrafchart file where level and temperature control has been implemented, following the procedure below. Note that there are several ways to implement the same behavior, where some ways are more efficient than others.

Preparation Step

Before the process can be operated with a continuous flow, the tank needs to be prepared. It is ready when the level and the temperature are at or slightly above the setpoint and when the agitator is turned on. This must be achieved in

several steps, since filling and warming will not be completed at the same time if they are started simultaneously.

When filling the tank, the inlet pump can be controlled directly by setting `InPump = F`. The value of F is already specified, and this should be used. The agitator is started by setting `AgitatorControlOn` to 1. Note that there is a built-in interlock that prevents the agitator and the heater to operate if the variable `LevelLow` in the workspace `LevelIndicator` is 1.

TASK 16
Determine a suitable way to bring the process to the desired state and create a procedure for this in the JGrafchart program, starting from the structure already available to the left in the main window. □

Main Step

The main step of the process is denoted `Run` in start.xml, and it is important that its named is not changed. This step will continue until `StopRun` becomes 1, which happens when the variable `RunTime` reaches the value `FinalTime` or when the user pushes the **Stop Run** button. In this step, the outflow should vary according to a predefined profile to simulate a variable demand from a downstream production step. The profile is started when the variable `OutFlowOn` is set to 1. The initial outflow is u_{out}^{-}, and after a while it is changed to u_{out}^{+}.

TASK 17
Add activation of the level and temperature controls in the main step. Note that the agitator should still be turned on. □

Finalization Step

When the main step has completed, the inflow should be turned off, the tank should be emptied and then cleaned in place. In this step we will have to extend the structure of `LevelIndicator` to also determine whether the tank is empty or full. For this purpose, a level indicator function is needed.

Level Indicator Function. The liquid level can vary continuously between 0 and 100 percent. A structure using boolean variables telling whether the tank is full or empty simplifies several other functions in the finalization step.

TASK 18
Enter the `LevelIndicator` workspace. It contains three boolean variables and a structure used to control the variable `LevelLow`, which is used to ensure that the heater and agitator cannot be on when the liquid level is too low. Create a new structure in this workspace with conditions and commands such that the following is fulfilled:

- `Full` should be 1 when `Level >= 70`, otherwise 0.
- `Empty` should be 1 when `Level <= 10`, otherwise 0.
- `LevelLow` should continue to work just like before. □

Emptying and Cleaning. When the level indicator has been implemented, we can continue the completion of the control sequence. The emptying of the tank should be performed by setting `OutPump` to a suitable value, for instance `F`. When the tank is empty (`LevelIndicator.Empty` is 1), then cleaning in place should be performed in the macro step named CIP. The macro step is opened in the same way as a workspace, and inside it, ordinary steps and transitions can be used to create a structure. In this macro step the cleaning procedure should be implemented as follows:

1. The tank should be filled with water. Use `LevelIndicator.Full` to determine when the tank is full.
2. The tank should then be emptied. Use `LevelIndicator.Empty` to determine when the tank is empty.
3. The tank is now clean and the control sequence should go back to the initial step.

TASK 19
Implement the finalization step according to the above specifications. □

10. Testing the Control Sequence

The complete sequence should now be ready for testing.

TASK 20
Start the sequence by pressing the **Start Sequence** button. Make sure that the correct commands are given at the right moments in time and that the process behaves according to the specifications. Use logging to store data that can be analyzed and plotted in MATLAB and then used in the report. □

11. Conclusions

In this project we have created a program for control of a CSTR process. The program contains both sequence control and PI control of level and temperature. Such a combination of discrete and continuous control is very common in practice and can be found in industrial processes as well as in household appliances such as washing machines.

TASK 21
Reflect upon the work that you have performed in this project. Discuss the modeling, the control and the sequence control; what worked well, what did not work so well, and how could the various parts be improved? What have you learned during the course of the project, both at a high level and at the detailed level? □

TASK 22
How can the project be improved to make it more instructive for future students? □

A. Introduction to JGrafchart

JGrafchart is a free software for development and execution of sequence control programs, developed at the Department of Automatic Control, Lund University. This appendix will only treat the subset of the JGrafchart functionality that is relevant to the project.

Example

A simple example of a JGrafchart program is shown in Figure 5. A lamp should be turned on when a button is pushed. The lamp should then stay lit for two minutes. The boolean variables `lamp` and `button` represent the lamp/button being

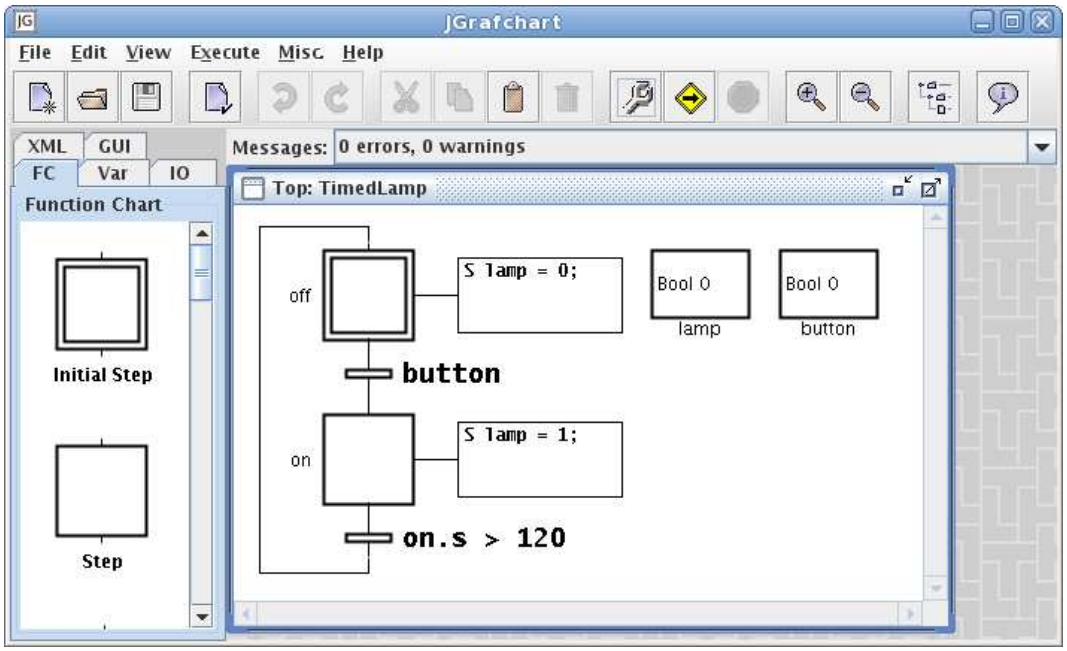


Figure 5 A simple JGrafchart program that turns on a lamp and then turns it off again after 120 seconds. Steps, transitions, and other elements are available in the palette to the left, while the program can be edited in the right window.

turned on/pushed, 1, or off/not pushed, 0. The two steps `off` and `on` corresponds to the lamp being turned on or not. When `button` becomes 1, the transition from `off` and `on` is taken, lighting the lamp. When the lamp has been lit for 120 seconds, the transition from `on` to `off` is taken and the lamp is turned off. The currently active step is marked with a black token when the program is run; this is not visible in the figure.

Programming in JGrafchart

Programming in JGrafchart is performed using drag and drop from the menu to the left into the workspace to the right. Objects can be marked, moved, deleted, cut, and pasted as in a typical graphical editor. Steps and transitions are connected by clicking and dragging on the short pins sticking out from the upper and lower parts of the objects. The standard rules of GRAFCET must be followed. For instance, it is not possible to connect two steps together without a transition inbetween.

Compilation

The program must be compiled before it can be run. This is done by pressing the *Compile* button in the upper toolbar or by choosing *Compile* in the *Execute*

menu. Compilation errors are shown in red in the *Messages* area just beneath the upper toolbar.

Two types of errors can occur at compilation time: syntax errors and semantic errors. For instance, the condition `y OR z` would generate a syntax error, since the correct syntax for an *or* expression in JGrafchart is `y | z` Semantic errors are generated for instance when undefined variables are referenced.

To run the compiled program, click the *Execute* button in the toolbar or select *Execute* in the *Execute* menu. Changes that have been performed after the latest compilation will not affect the program being executed.

Execution

A JGrafchart program is executed periodically. Each cycle, the following operations take place:

1. All digital and analog inputs are read.
2. All transitions whose conditions are true are marked.
3. All marked transitions are activated, deactivating the preceding steps and activating the following steps (see Actions below).
4. All step clocks (accessed by `<stepName>.t`) are updated.
5. All variables that are affected by step actions are updated, including the writing of digital and analog outputs.

Syntax

The syntax for expressions in JGrafchart is quite similar to that of Java. An important difference is that `0` and `1` are used as boolean constants (rather than `false` and `true`).

The following operators are supported by JGrafchart: `+`, `-`, `*`, `/`, `!` (negation), `&` (and), `|` (or), `==` (equality), `!=` (inequality), `<`, `>`, `<=`, `>=`.

Variable names are local to each workspace of the program. For instance, `Y` refers to the variable `Y` of the current workspace. Access to variables in other workspaces can be done using dot notation. For instance, `W2.Y` refers to the variable `Y` in the workspace `W2`.

The expression `stepName.x` returns `1` if the step is active and `0` otherwise, while `stepName.t` returns the number of cycles that the step has been active (`0` if the step is not active).

Actions

For each step, a number of different *actions* can be specified, that should be executed when the step is activated, active, and deactivated. Four different types of actions are supported in JGrafchart:

- **Start action (S)**: An action that is taken when the step is activated.
  `S <action>;`
- **Periodic action (P)**: An action that is executed once per cycle, as long as the step is active.
  `P <action>;`
- **Exit action (X)**: An action that is taken when the step is deactivated.
  `X <action>;`

- **Normal/boolean action (N)**: Connects a boolean variable to the state of the step. The variable becomes 1 when the step is activated and becomes 0 when the step is deactivated.
  `N <boolean variable>;`

`<action>` could either be an assignment or a method call:

- **Assignment**: `<variable> = <expression>`
- **Method call**: `<method>()`

Note that an action must always be terminated by a semicolon!

Transition Conditions

Each transition has a condition that should evaluate to either `0` or `1`. If the condition is `0` then the transition will not happen, while `1` will enable a transition if the preceding step is active.

Building Blocks in JGrafchart

Steps. The name of the step is shown to the left, and the actions are shown in the action box to the right of the step. Right-clicking on the step makes it possible to hide/show the action box, edit the actions, or change the name of the step.

Initial Step. All initial steps are activated when the execution of the program starts.

Transitions. A transition is associated with a condition that must be true, `1`, to enable a transition between two steps.

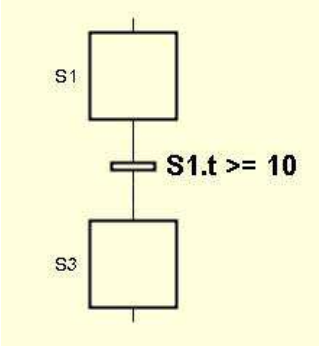


Figure 6 A transition.

Macro Steps. A macro step represents a hierarchical abstraction containing its own workspace, see Figure 7. The workspace is opened or closed by right-clicking on the macro step and selecting `Show/Hide Body`. The first step inside a macro step is an enter step, where the execution will start. Similarly, the last step is an an exit step, where the execution finishes. Besides being marked with arrows, the enter and exit steps are ordinary steps that have the same menu choices as regular steps. Using macro steps, it is possible to divide a large program into smaller, manageable pieces.

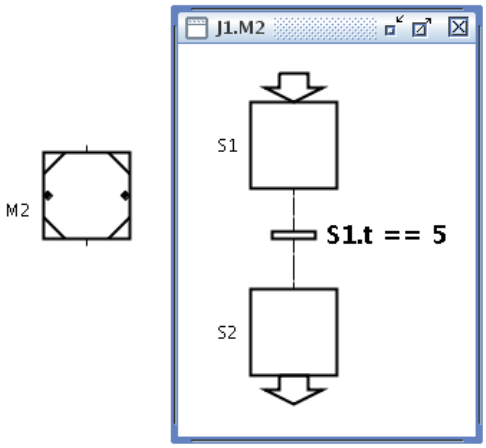


Figure 7 A macro step M2 and its associated workspace, containing the enter step S1, the exit step S2, and a transition inbetween.

Variables. A variable in JGrafchart can be of the type (*real, boolean, integer* or *string* and always have an associated name and value. Variables are graphically represented by boxes, where the name of the variable is found under the box and the

Action Buttons. An action button performs a specific action when you click on it.

Workspace Objects. A workspace object creates a separate part of the program and is used to structure the code. The PI controllers in this project are implemented in separate workspaces.

Text Objects. Text objects can be used as comments in the program.

Documentation

For more detailed documentation, it is recommended that you on the *Online Help*, found in the *Help* menu. You can also get more information about a given object by first selecting *Object Help* in the toolbar and then clicking on an object.

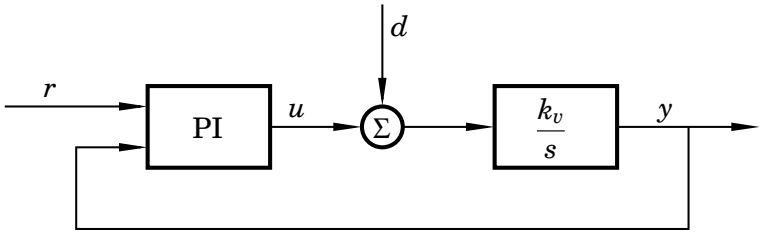


Figure 8 Block diagram for PI control of an integrating process.

B. Arrest Time Tuning

Arrest time tuning is a simple design method for PI controllers, aimed at load disturbance rejection in processes where the transfer function from input to output can be viewed as a pure integrator:

$$Y(s) = \frac{k_v}{s} U(s) \tag{7}$$

Control of an integrating process subject to a constant load disturbance is illustrated in Figure 8. The arrest time, T_a, is defined as the time it takes for the process output to turn back to the setpoint after a step disturbance at the process input, d. This is illustrated in Figure 9. It can be shown that the regulator parameters that are needed to achieve a desired value of T_a are given by

$$K = \frac{2}{k_v T_a}, \tag{8}$$

$$T_i = 2T_a. \tag{9}$$

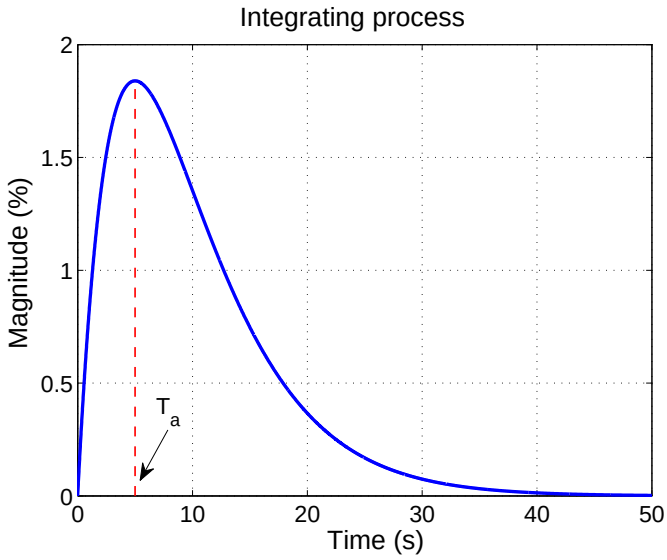


Figure 9 The arrest time T_a is the time it takes before the process output turns back towards the setpoint after a step load disturbance at the process input.