A Freely Available Interactive PID Learning Module

Theorin, Alfred; Johnsson, Charlotta

2014

A Freely Available Interactive PID Learning Module ⋆

Alfred Theorin * **Charlotta Johnsson** †

* *Department of Automatic Control, Lund University, Lund, Sweden (e-mail: alfred.theorin@control.lth.se)*
† *Department of Automatic Control, Lund University, Lund, Sweden (e-mail: charlotta.johnsson@control.lth.se)*

The PID is by far the most commonly used controller in industry. There are billions of control loops and PIDs are used for more than 95% of them (Åström and Murray (2012)). The PID controller is taught in most introductory automatic control courses and given its wide use it is important that the students obtain a deep understanding for PID control.

Students often prefer either experiments or abstract concepts as starting point for constructing new knowledge. Since this is individual, both starting points should be provided. Interactive learning tools is a great way to give students the opportunity to get hands on experience. Examples of interactive learning tools are ICtools (Johansson et al. (1998)) and Interactive Learning Modules for PID (Guzman et al. (2008)). ICtools is a MATLAB based tool for automatic control basics whereas Interactive Learning Modules for PID covers many aspects of PID controller design and tuning.

Students should also be familiarized with PID controllers in a realistic setting, that is, controlling a process. Students can then get an intuitive feel for PID control and tuning. For a PID learning tool to be complete it should include all common PID features: Anti-windup, Auto/Manual mode, Bumpless mode change (Auto/Manual), Bumpless parameter change, Feedforward, Maximum derivative gain, Setpoint weighting, and Tracking.

Grafchart is based on Sequential Function Charts (SFC), one of the IEC 61131-3 PLC standard languages which is used to implement sequential, parallel, and general state-transition oriented applications. Grafchart has the same graphical syntax as SFC with steps and transitions where steps represent possible application states and transitions represent change of application state. Associated with the steps are actions which specify what to do. Associated with each transition is a Boolean guard condition.

Grafchart extends SFC with hierarchical structuring, reusable procedures, and exception handling to make it convenient to implement large applications. Reusable code can be put in a Procedure which can then be called from Procedure Steps and Process Steps.

Grafchart applications are, like SFC, executed periodically, one scan cycle at a time. Since Grafchart is a state oriented language and not a data flow language, ensuring the desired execution order requires knowledge about its execution model:

1. Read inputs.
2. Fire transitions (`X` and `S` actions are executed).
3. Execute `P` actions.
4. Update variables subject to `N` actions.

JGrafchart is a freely available development environment for Grafchart with some extensions. The most interesting extensions for this paper are *inline if* and graphical elements. *Inline if* is used for conditional expression evaluation and can make applications easier by reducing the number of steps. The syntax is the same as in C and Java: `<cond>?<trueExpr>:<falseExpr>`. The graphical elements are among others rectangles, ellipses, lines, arrows, text fields, images, lists, and plots. They can be accessed from step actions to create animations and operator interfaces. There are also buttons with associated *on click* actions. Variables are shown with their current value which can be edited during execution. Here this is particularly useful for controller parameters and modes. JGrafchart can also connect to external environments through customizable inputs/outputs (I/O) to control physical processes.

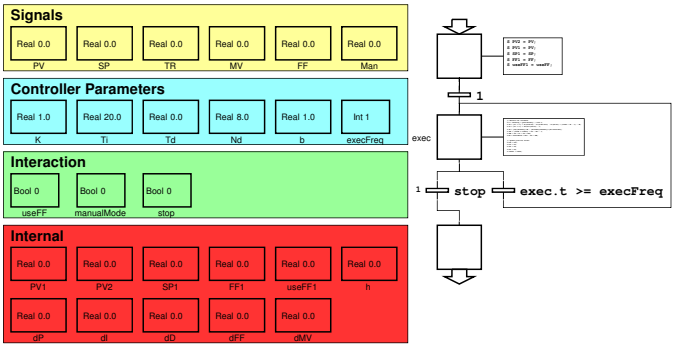

Fig. 1. The new `PID` Procedure. Step actions are not intended to be readable in this figure.

A full-fledged PID module has been implemented in JGrafchart. It is implemented on velocity form which means that it calculates control signal increments. The velocity form has fewer internal states and requires fewer special cases which means less code, specifically considerably less rarely executed special case code which is more likely to contain errors.

⋆ Financial support from VINNOVA and the Swedish Research Council through the LCCC Linnaeus Center grant, the ELLIIT Excellence Center, and the LISA project are gratefully acknowledged.

It is implemented as a Procedure, see Fig. 1, and is thus reusable. It is called `PID` and its parameters are explained in Table 1. The PID algorithm is implemented in a single step and can thus execute at the same rate as the caller. The code is shown in Fig. 2 and is a straightforward implementation of the discrete equations for a PID on velocity form, see (Åström and Hägglund (2006)).

Table 1. `PID` Procedure parameters.

Parameter	Description
PV	Process Value, measurement value of the process.
SP	SetPoint, the reference value.
TR	TRacking signal, the last actuated control signal.
MV	Manipulated Variable, the control signal.
FF	FeedForward, a signal added to the control signal.
Man	Manual, the control signal when in manual mode.
K	Controller gain.
Ti	Integral time.
Td	Derivative time.
Nd	Maximum derivative gain (usually called N).
b	Setpoint weight for the P-part.
execFreq	Number of caller scan cycles for a PID sample.
useFF	Signal to turn feedforward on/off.
manualMode	Signal to turn manual mode on/off.
stop	Signal to terminate the Procedure call.

```
// Execute one increment
S h = execFreq * getTickTime() / 1000.0;
S dP = (Ti != 0) ?
         K*(b*SP-PV) - K*(b*SP1-PV1) :
         K*(SP-PV) + (useFF ? FF : 0) - TR;
S dI = (Ti != 0) ? K*h/Ti*(SP-PV) : 0;
S dD = (Td/(Td+Nd*h))*dD -
         (K*Td*Nd/(Td+Nd*h))*(PV-2*PV1+PV2);
S dFF = (useFF & useFF1) ? FF - FF1 : 0;
S dMV = dP + dI + dD + dFF;
S MV = manualMode ? Man : TR + dMV;

// Update previous values
S PV2 = PV1;
S PV1 = PV;
S SP1 = SP;
S FF1 = FF;
S useFF1 = useFF;
```

Fig. 2. The main step code of the `PID` Procedure.

Procedure parameters can be *call-by-reference* (`R`), *call-by-value* (`V`), or *default* (omitted). For *call-by-value* and *default* the parameter is set once when the call is made. For *call-by-reference* the parameter is tied to a variable or I/O in the calling context. To be useful, `PV`, `SP`, `TR`, and `MV` should be *call-by-reference* and the caller should set and update all parameters except `MV` which is the `PID` Procedure's sole output.

An example call is shown in Fig. 3. Here parameters for feedforward, manual mode, setpoint weighting, the D-part, and stopping are omitted and will thus get their *default* values, which means these features are not used. *Call-by-reference* is used for all parameters, for example the parameter `PV` is a reference to `Level` in the caller context.

```
R PV = Level;
R SP = LevelRef;
R TR = InPump;
R MV = InPump;
R K = K;
R Ti = Ti;
```

Fig. 3. Parameters for an example `PID` Procedure call.

A tricky part of setting up a simulated process that uses the `PID` Procedure is to ensure proper execution order:

1. Execute the PID controller.
2. Limit the control signal.
3. Update the simulated process.

The `PID` Procedure uses `S` actions to make it execute as early as possible (execution model phase 2). Hence `P` actions can be used for the simulated process (execution model phase 3). To ensure that the limited control signal is used to update the simulated process, the limiting should be a preceeding `P` action in the same step.

The PID module has been evaluated on a simulated water tank process, see Fig. 4. The inflow is controlled with a pump, there is an outflow through a hole, and the objective is to control the level. Live animation and plots show the state of the process and both setpoint and basic control parameters can be changed during execution. This is a suitable setup for beginners. More of the `PID` Procedure parameters can be included in subsequent exercises.

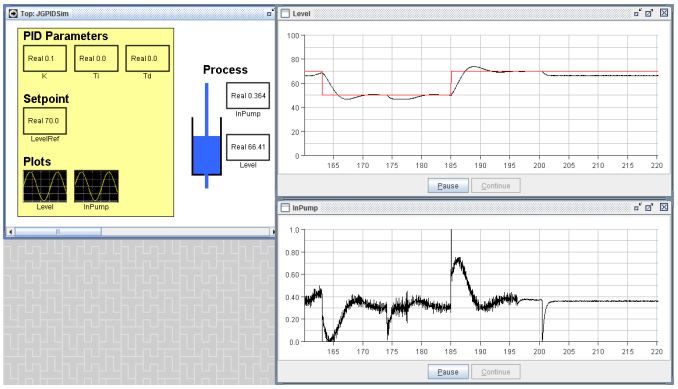

Fig. 4. In the left part, setpoint and controller parameters can be changed and an animation of the process is shown. The upper plot shows measurement value (black) and setpoint (red). The lower plot shows the control signal.

Educational tools should be easily available for everyone. The interactive PID module and sample applications will be part of future JGrafchart releases. JGrafchart is a Java application that is platform independent and freely available for download from:

http://www.control.lth.se/Research/tools/grafchart.html

REFERENCES

Åström, K. and Hägglund, T. (2006). *Advanced PID Control*. ISA.

Åström, K. and Murray, R. (2012). *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, Princeton and Oxford. URL http://www.cds.caltech.edu/~murray/amwiki.

Guzman, J.L., Åstrom, K., Dormido, S., Hägglund, T., Berenguel, M., and Piguet, Y. (2008). Interactive learning modules for PID control [lecture notes]. *Control Systems, IEEE*, 28(5), 118–134.

Johansson, M., Åström, K.J., and Gäfvert, M. (1998). Interactive tools for education in automatic control. *IEEE Control Systems*, 18(3), 33–40.