10.2478/v10170-010-0039-y

Archives of Control Sciences
Volume 21(LVII), 2011
No. 2, *pages 189–205*

Self-tuning run-time reconfigurable PID controller

MARIUSZ PELC

Digital PID control algorithm is one of the most commonly used algorithms in the control systems area. This algorithm is very well known, it is simple, easily implementable in the computer control systems and most of all its operation is very predictable. Thus PID control has got well known impact on the control system behavior. However, in its simple form the controller have no reconfiguration support. In a case of the controlled system substantial changes (or the whole control environment, in the wider aspect, for example if the disturbances characteristics would change) it is not possible to make the PID controller robust enough. In this paper a new structure of digital PID controller is proposed, where the policy-based computing is used to equip the controller with the ability to adjust it's behavior according to the environmental changes. Application to the electro-oil evaporator which is a part of distillation installation is used to show the new controller structure in operation.

Key words: policy-based computing, reconfigurable systems, PID controller

1. Introduction

Digital PID controllers constitute an overwhelming group if typical industrial applications of the computer control systems of all kinds are considered. It is because of inherent advantages of PID control algorithms which are mainly as follows:

- behavior of the control system with PID controller is easily predictable,
- there exist number of convenient and simple tuning methods for PID controllers which take into account various optimization criteria,
- software implementation of digital PID controller is simple and not resource–consuming so the controller can be implemented even in a very resource constrained embedded systems.

Most of the tuning methods, especially basing on Ziegler-Nichols approach, can be used only in an off-line mode and cannot be easily adapted for an on-line (real-time) application. As the consequence, self-tuning functions of PID controllers need separation of the tuning cycles (typically iterative and thus time consuming) from the control cycles.

The Author is with Opole University of Technology, Faculty of Electrical Engineering, Automatic Control and Informatics, ul. Mikolajczyka 5, 45-272 Opole, Poland. E-mail: m.pelc@po.opole.pl

Received 23.04.2011.

In many real-time system applications, this approach may cause serious deadline-related problems and ultimately may lead to a system failure.

This situation makes a space for development of new tuning algorithms which would provide some special features that follow requirements concerning real-time support of the self-tuning. These requirements refer to a total transparency from the point of view of the control algorithm and background operation as long as the tuning procedure are not finished in order not to affect the controlled process. Besides, in the contrary to the typical self-tuning algorithms the ideal tuning algorithm is also expected to take into account not only the parameters which are strictly related to the controlled system (related to its mathematical model or its parameters), but also some other factors, which have direct impact on the control process (such as system noise, disturbances, etc.). Furthermore, as the choice of tuning method usually influences achievable type of optimality, it would be highly desired that the core part of the tuning algorithm – the tuning logic – was replaceable so that the tuning process itself was flexible and able to satisfy different and sometimes contradicting tuning goals (optimality in the meaning of integral-square-error criteria, time optimality, etc.).

Typical solution of the self-tuning/reconfiguration problem uses artificial intelligence methods, such as Artificial Neural Networks, Fuzzy Logic, Evolutionary Algorithms, etc. In this paper alternative approach is proposed namely policy-based computing. It is used for the purpose of implementing self-tuning, reconfiguration and context-awareness features into digital PID controller. This technology is accompanied with the Open Decision Point component architecture [20]. The technology seems to have most of the required features (if not all of them) of the ideal tuning technology. Moreover, it is free from typical drawbacks of Artificial Neural Networks, Fuzzy Logic or Evolutionary Algorithms. This technology offers simple reconfiguration (both, structural – by enabling/disabling certain policy elements, and syntactical – by supporting range checks and utility functions) and supports expert-knowledge-based decision making.

2. Related works

The idea of embedding of self-tuning capabilities into PID controller is not entirely new and has been exploited over the last two decades. Modern computer control systems, however, are much more complex in comparison to those used few years before. This follows mainly from the new technologies and application domains. In such applications, the self-tuning capability and context-awareness became crucial from the point of view of satisfying of some run-time changing control goals. These domains cover industrial processes control, autonomic systems and robotics. There is a strong demand for self-tuning algorithms to be able to self-tune the controller at the pre-deployment stage but also in the run-time. A technology that would add to this also a possibility of post-deployment reconfiguration of the self-tuning procedure itself is even more advantageous.

In the case of Fuzzy Logic, reasoning is applied to the process of self-tuning PID controllers, and the solution, in general, goes in the following two directions:

- Fuzzy Logic system is used for tuning the PID controller gains,
- Fuzzy Logic system performs the PID controller function.

In [16] the self-tuning algorithm for PID controllers based on the theory of adaptive interaction is presented. The advantage of this algorithm is mostly its high robustness resulting from the fact, that no prior knowledge about the controlled system is required. This makes the control algorithm robust to the changes of the system and enables its application to linear and non-linear systems. The algorithm incorporates a mathematical tools to calculate the adequate values of PID controller gains based on the solution of the task of control error minimization.

The idea of Fuzzy PID controller (FPID) is presented in [13]. A typical Mamdani Model with triangle-shaped membership functions for linguistic variables and center of gravity defuzzification method is used. The outputs of Fuzzy Parameter Regulator are used for direct scaling of the PID controller's gains. The base for the Fuzzy Parameter Regulator consists of 25 rules. Adjustments/tuning of the control algorithm uses information provided by relative rate observer.

Similar approach based on Fuzzy Mamdani Model with modified membership functions for linguistic variables is presented in [21] in the application to the induction motor control system and in [19] where the fuzzy system is responsible for run-time adjustments of PID gains (pre-calculated using Ziegler-Nichols method) in the task of temperature control in metal chamber.

In [12] another application of fuzzy PID controller is described. The application concerns the unmanned aircraft control. In this application, the self-tuning feature is the only way to guarantee the best dynamic performance for a wide variation of system parameters changes in run-time (during flight). This constitutes the main advantage of the application over the typical control systems designed in a traditional way (pre-deployment, based on assumed fixed parameters). Again, Fuzzy Mamdani Model is used for calculation of the optimal PID gains (from the point of view of the control goal).

As mentioned above, the self-tuning PID algorithms can also be implemented as Fuzzy System of an adequate structure and logic. In this kind of self-tuning algorithm the Fuzzy System is not used to tune the PID controller but *is* a PID controller itself. This solution is described in [14] where the Fuzzy PID controller is used in the active control of magnetic bearing. The control task is to regulate the disturbances and suppress the unbalancing vibration of the rotor. Fuzzy-Neuro solutions inherit actually both mentioned problems. Very similar solution with the application to the power system stabilization is presented in [10].

Alternatively to Fuzzy Systems, Artificial Neural Networks (ANNs) can be used to implement the self-tuning features of the PID controller, as well as hybrid Neuro-Fuzzy or Neuro-Genetic algorithms or even evolutionary algorithms (for example genetic algorithms). Similarly to the Fuzzy Systems, there are various approaches to solve the task of tuning the PID controller gains.

An example application of Neuro-PID controller is described in [11]. This controller is used in the typical task of double inverted pendulum stabilization. The gains of the PID controller are tuned using a Neural Network in this case. Another application uses Neuro-PID controller to the task of tracking control of a discharge air temperature system ([23]). Neuro-PID controller is used also to control a Power Plant [22] as an example MIMO (Multi Input Multi Output) system.

A hybrid Neuro-Genetic approach to the problem of self-tuning of the PID controller is presented in [15]. In this solution the controlled systems are identified by the Neural Auto-tuner returning appropriately scaled PID controller gains. The genetic algorithm is responsible for optimization of a weighted cost function (this function optimizes the PID gains returned by neuro auto-tuner). In [9] a genetic algorithm (GA) is used for PID parameters tuning whereas the Neural Network is used as the plant model. In this solution the PID gains are calculated off-line and then used on-line for the control purposes.

The above summary proves that there are number of solutions which are used to obtain a self-tuning PID controller. However, the alternative solution proposed in this work based on policy-based computing has some important advantages which make it much more flexible than the other approaches. This most important advantages are: support for utility functions, run-time self-reconfiguration of policies, post-deployment re-configuration, and context-awareness.

3. AGILE policy definition language

As the solution which can be the alternative for the typical approaches exploring Fuzzy Logic, Artificial Neural Networks and Genetic Algorithms for the purpose of widely understood diagnosis, in this paper the AGILE Policy Definition Language (PDL) is used to define various control/supervision policies. This language specification was precisely described in number of works (e.g. [2], [4], [3]). However, for the purpose of this paper, the most important elements of the AGILE need to be recalled, to make it entirely clear how the AGILE PDL may fit to the typical problems related to control systems. This language is based on XML structure and defines the following policy objects:

- *PolicySuite* contains all policy objects.
- *Policy* defines decision making policy (one source policy file may contain zero or more policies). Policies can load *Templates* or execute *Actions*. Example of *Policy* is shown below.

```
<Policy Name="Policy1"
 PolicyType="NormalPolicy">
  <Load Template="Template1"/>
  <Execute Action="Action1"/>
</Policy>
```

- *Template* can be used to configure policies (for example, to assign some values to *InternalVariabies*). In this way it provides customization of policies as different users can have different preferences to be loaded into current policy. Example of *Template* is shown below.

```
<Template Name="Template1">
  <Assign Variable="InternalVariable1"
   Value="7"/>
</Template>
```

- *Action* is the execution element of a policy and it gathers policy decision-making logic (it evaluates *Rules*, *TRCs*, *UFs*, it can also set local variables or yield *Policies*) and finally returns a policy decision. Example of *Action* is shown below.

```
<Action Name="Action1">
  <Assign LHS="InternalVariable1"
   RHS="true"/>
  <EvaluateRule Rule="Rule1"/>
  <Yield Policy="Policy2"/>
</Action>
```

- *Rule* is used to compare variables (*ExternalVariables* or *LocalVariables*), values, etc. Depending on the comparison result an appropriate *Action* can be executed. Example of *Rule* structure is presented below.

```
<Rule Name="Rule1"
 LHS="EnvironmentVariable1" Op="EQ"
 RHS="true" ActionIfTrue="Allow"
 ElseAction="Block"/>
```

- *ToleranceRangeCheck* (TRC) is an implementation of dead-zone which is especially useful in case of tracking of a dynamic goal (or to check how an interesting parameter behaves); the dead-zone is specified by *Tolerance* TRC parameter. Depending on the value of the tracked parameter an appropriate action is executed. Example of *TRC* structure is presented below.

```
<TRC Name="TRC1"
 Check="SampleEnvironmentVariable1"
 Compare="SampleInternalVariable1"
 Tolerance="5" ActionInZone="null"
 ActLower="Action1" ActHigher="Action2"/>
```

- *UtilityFunction* (UF) is a very sophisticated policy object mostly used to indicate goal-attainment or want-satisfaction. Depending on the utility level an appropriate *Action* is taken. Example of *UtilityFunction* is shown below.

```
<UF Name="SampleUF" Terms="2">
  <Option Action="Action1"
   T1="SampleInternalVariable1"
   W1="1" T2="3" W2="4"/>
  <Option Action="Action2"
   T1="SampleEnvironmentVariable1"
   W1="5" T2="10" W2="2"/>
</UF>
```

4. Digital PID controller

A model of an ideal continuous PID controller can be expressed as:

$$u(t) = K\left[e(t) + \frac{1}{T_i}\int_0^t e(t)d\tau + T_d\frac{de(t)}{dt}\right] \tag{1}$$

The controller described by (1) cannot be implemented in the computer control system due to the integration window while $t \to \infty$. So in practice an approximation of the integration part (e.g. rectangular one). The result of the control signal in the k-th sampling period is of form:

$$u(kT_0) = Ke(kT_0) + K\frac{T_0}{T_i}\sum_{i=0}^{k-1} e(iT_0) + +K\frac{T_d}{T_0}\left(e(kT_0) - e((k-1)T_0)\right) \tag{2}$$

PID controler described by (2) is sometimes called a *position PID algorithm* and the control signal is calculated in each control cycle. After subtraction from the equation (2) the equation:

$$u((k-1)T_0) = Ke((k-1)T_0) + K\frac{T_0}{T_i}\sum_{i=0}^{k-2} e(iT_0) + K\frac{T_d}{T_0}\left(e((k-1)T_0) - e((k-2)T_0)\right) \tag{3}$$

one can get the final form of digital PID controller given by:

$$u(kT_0) = u((k-1)T_0) + q_0e(kT_0) + q_1e((k-1)T_0) + q_2e((k-2)T_0) \tag{4}$$

where:

$$q_0 = K\left[1 + \frac{T_d}{T_0}\right], q_1 = -K\left[1 + 2\frac{T_d}{T_0} - \frac{T_0}{T_i}\right], q_2 = K\frac{T_d}{T_0}$$

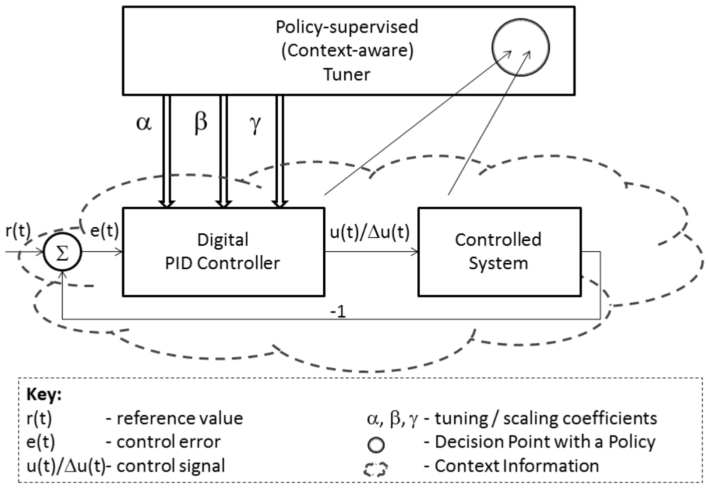


Figure 1. Structure of self-tuning digital PID controller.

The PID controller described by (4) is also called *incremental PID algorithm* because in each control cycle a change of control signal

$$\Delta u = u(kT_0) - u((k-1)T_0)$$

is calculated instead of the actual control signal.

5. Self-tuning and reconfiguration of digital PID controller

The architecture of Self-tuning Context-aware Digital PID controller is shown in Fig.1. It comprises of two main components:

- A Policy-supervised Context-aware Tuner (PCT) which is tuning the PID controller based on the currently available contextual information,
- A Digital PID controller with variable gains tuned by the PCT.

The PCT is policy-configurable because it implements the Open Decision Point architecture described in details in [20] and in [4]. Policies use current context information (reflecting the whole control system environment) to calculate the most appropriate scaling coefficients α, β and γ or to perform some structural changes within the policy, in order to make it better suit to the formulated control goals. The Open Decision Point architecture supports run-time policy replacement without the need to re-deploy the whole control component. Thus the new architecture of digital PID controller has self-tuning, context-awareness and run-time reconfiguration features.

6. Policy-supervised tuning

The self-tuning capability of the PID controller is achieved purely based on policies flexibility. In every control/calculation cycle the PCT is updated with the information about the exact system/environment state. The information about the system state can be provided by a set of sensors (directly) or alternatively, part of the information can be obtained for example from Exact State Observers (ESOs) described in details in [6], [5], [7] or [8]. Once the PCT has all the information needed for making decision, it evaluates the currently loaded control policy in order to decide what values of gains scaling coefficients are the most appropriate in the given environmental circumstances. The gains values are the outcome of the control policy.

Depending on the used Policy Definition/Description Language (PDL), the task of tuning of the PID controller can be achieved in different ways. In the case of AGILE policies ([2], [1], [17]) which are used in this work, the simplest way to achieving the self-tuning capability by PID controller is to design a policy which operates as a very sophisticated gain scheduler and selects the most appropriate scaling coefficients for the PID gains from an expertly defined set.

Alternatively, policy can return directly the most suitable gains of the PID controller (obtained for example by the use of Ziegler-Nichols method). These solutions would also be the least time consuming because the decision making logic would be constituted as a set of rules (similarly to Fuzzy Logic Systems). These rules define ranges of context parameters which gains (or scaling coefficients) should be selected. In this way the equations (1)-(4) will remain the same, but $\overline{K}$, $\overline{T_d}$ and $\overline{T_i}$ gains would be used instead of K, T_d and T_i gains, where:

$$\begin{aligned}\overline{K} &= K\alpha,\\ \overline{T_d} &= T_d\beta,\\ \overline{T_i} &= T_i\gamma.\end{aligned}$$

and coefficients α, β, γ constitute the tuning policy outcome which is used to scale the PID controller gains.

In the case of more sophisticated way of choosing the best PID gains, it would be simply required to provide a more complex AGILE policy and replace the above one. This includes support for *Utility Functions* or *ToleranceRahge checks* and additionally the ability of the policy to change its own structure in run-time.

As mentioned above, the *Utility Functions* allow to specify (via selection of the utility function form or by changing some weight coefficients) which utility function is the best to satisfy the set of control goals in the given environmental circumstances.

AGILE policies allow also the policy designer to develop a policy of the structure change during evaluation in response to the environment changes. This is possible because all policy objects that are responsible for decision making are *guarded*. Thus these policy objects are accessible (active) only if guards permit to do so. Otherwise the policy objects are inactive, changing in this way the policy structure in run-time.

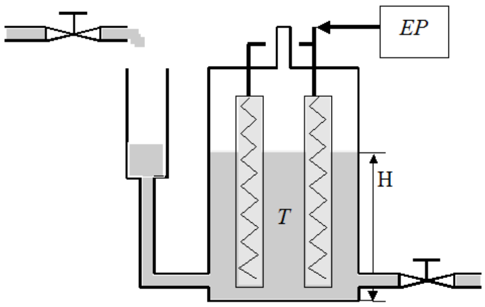


Figure 2. Electric-oil evaporator

7. Simplified model of electric-oil evaporator

The example of a system used for the numerical tests is an electric-oil evaporator shown in Fig.2. The electric-oil evaporator is a part of a chemical system used for distillation of a binary mixture (details can be found in [18]). For the control purpose, a mathematical model describing the controlled system was identified. As the original (real) mathematical model characteristic was not very suitable for simulations (due to very long time constants which would imply very long simulation times), its normalized form was used for tests in this work. The transfer function (in Laplace form) describing the electric-oil evaporator in the normalized form is given by:

$$\frac{T(s)}{EP(s)} = \frac{0.9}{s^3 + 3.8157s^2 + 3.9311s + 1} \tag{5}$$

where:

T	- is the binary mixture temperature,
EP	- electric power used to heat the mixture.

The main control task is optimization of the distillation process efficiency by controlling the binary mixture temperature with respect to the power usage.

8. Simulations

For the simulation purposes the Ziegler-Nichols method was used to determine the critical gain and oscillation period for the considered evaporator model. The following values were obtained:

$$\begin{aligned} K_{cr} &= 15.6 \\ T_{cr} &= 3.2 \end{aligned}$$

Hence the PID controller gains were initially set as :

$$K_p = 0.6K_{cr} = 9.4 \tag{6}$$

$$T_i = 2K_p/T_{cr} = 5.8 \tag{7}$$

$$T_d = K_pT_{cr}/8 = 3.7 \tag{8}$$

In order to show that the digital PID controller may be controlled by a policy (and thus can have ability of self-tuning, reconfiguration and context-awareness), a very simple case was designed showing temperature control application using a simple control policy. This policy uses a TRC (Tolerance Range Check) in order to make decisions regarding the controller tuning. The policy can be found in Appendix. The decision making part of the policy using a *ToleranceRangeCheck* is shown in Fig.3. The policy (see Ap-

```
<ToleranceRangeChecks>
  <TRC Name="ErrorCheck"
   Check="ReferenceError"
   Compare="ActualError" Tolerance="2"
   ActionInZone="Mode2"
   ActionHigher="Mode1"
   ActionLower="Mode3"/>
<ToleranceRangeChecks/>
```

Figure 3. Decision-making *ToleranceRangeCheck*

pendix) defines three output variables/coefficients: α, β and γ. These variables are used for scaling the initial PID controller gains: K_p, T_i and T_d, respectively . The policy compares the current control error denoted in policy by *ControlError* (difference between reference value and the controlled variable) with a reference value of error denoted by *ReferenceError*. Depending on what is the current difference between the value of the control error and the reference value (expressed in %), the policy switches the PID controller into one of three available modes: *Mode1*, *Mode2* and *Mode3*. In each of the modes the values of α, β and γ variables are different. To be exact, in this example, the extent of changes was limited with respect to β coefficient only for clarity sake (to simplify the policy analysis). However, by changing the other coefficient accordingly, a policy designer could achieve even better results.

The described policy was used for simulations presented in this section. The simulation scenario was very simple as its main purpose was to show applicability of policy-based computing to the task of PID tuning rather than to focus on achieving some spectacular tuning results. The simulations covered the following three areas:

- simulation of a system with PID controller tuned only with Ziegler-Nichols method and calculation the integral square error value,
- simulation of a system with PID controller with policy tuning the Ziegler-Nichols gains appropriately and calculation the integral square error value,

- comparison of the integral square error values for both systems showing the differences in control quality.

The simulation results are shown in Fig.4 and in Fig.5 The simulation time was 50s and

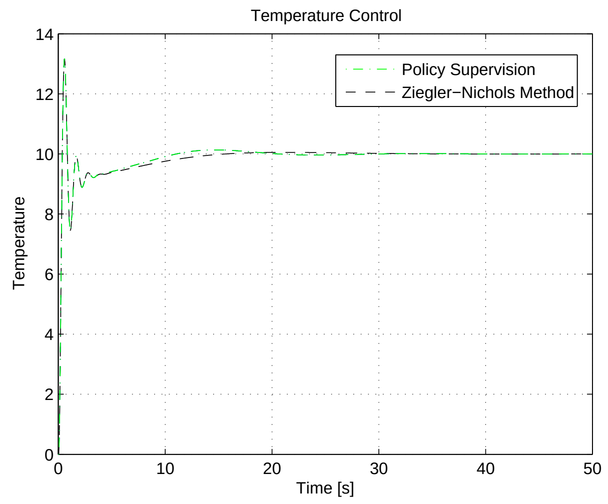


Figure 4. Temperature Control

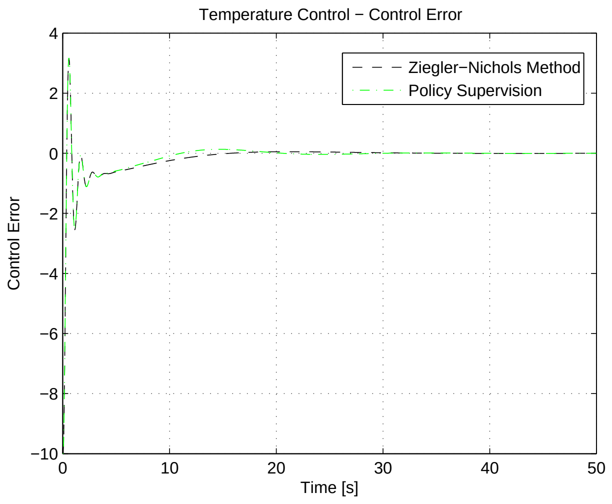


Figure 5. Control Error

the simulation step was 0.01s. The values of integral square error in the system with and without policy supervision were the following:

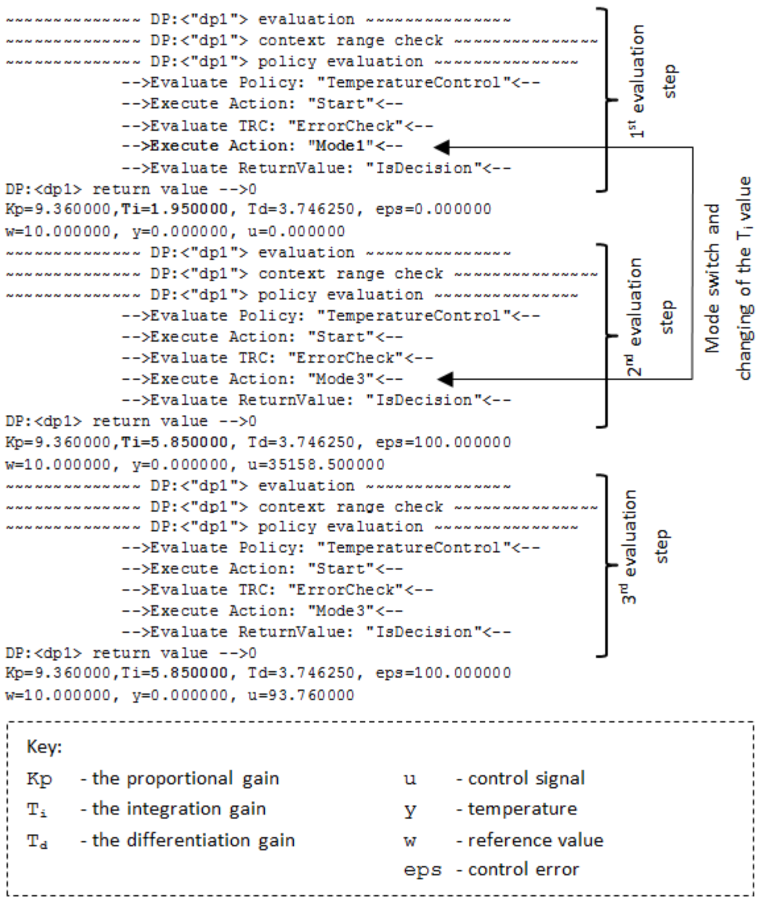
$$\left(\int_0^T e^2(t)dt\right)_{Z-N} = 25.83261,$$
$$\left(\int_0^T e^2(t)dt\right)_{POL} = 25.47897.$$

Based on the simulation results shown in Fig.4 and in Fig.5 and after comparison of the two integral square error values one can see that there is a slight difference between the way the PID controllers work with/without policy supervision (policy-supervised system proved to guarantee a bit better control quality). Although this difference is small (as mentioned before, it was not the main goal of the simulations), it clearly shows that policies can be successfully used for tuning PID controllers. Besides, the policy used for simulations was very simple and actually limited the extent of PID gains change to change of K_p only. In case of more sophisticated policy (with more TRCs or with UFs applied), the results would be even more convincing. What is the most worth to emphasize is that in the case of the control system with PID controller is tuned using fixed Ziegler-Nichols gains, the controller does not have any capability to adjust its own behavior in response to the environment changes. In the contrary, the policy supervised system offers high flexibility because, first of all, the policy can be tuned (to serve at best its purpose) and secondly, policies can be replaced in run-time (at any time) with a newer version changing/improving/adjusting the control strategies depending on applications/circumstances etc.

A part of the real-time simulation trace is shown in Fig.6. As one can see in Fig.6, the policy decision was to increase the value of T_i gain from $T_i = 1.95$ to $T_i = 5.85$. This resulted in relative increase of the control signal value (in comparison to the reference value of the control signal calculated in the system without policies) as its integral component increases. In this way the PID controller may react in a more effective way as the convergence of the controlled variable (temperature) to the reference value will be faster.

9. Conclusions

In this paper a new application of AGILE policies to the task of PID digital controller tuning was presented. An architecture of the hierarchical system with Policy Context-aware Tuner was presented and verified in the simulation way. Though a very simple reference criterion was used to compare traditional control algorithm with PID controller working with fixed settings (Ziegler-Nichols) and the new type of controller, the results



```
~~~~~~~~~~~~~~ DP:<"dp1"> evaluation ~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~ DP:<"dp1"> context range check ~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~ DP:<"dp1"> policy evaluation ~~~~~~~~~~~~~~~
            -->Evaluate Policy: "TemperatureControl"<--
            -->Execute Action: "Start"<--
            -->Evaluate TRC: "ErrorCheck"<--
            -->Execute Action: "Mode1"<--
            -->Evaluate ReturnValue: "IsDecision"<--
DP:<dp1> return value -->0
Kp=9.360000,Ti=1.950000, Td=3.746250, eps=0.000000
w=10.000000, y=0.000000, u=0.000000
~~~~~~~~~~~~~~ DP:<"dp1"> evaluation ~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~ DP:<"dp1"> context range check ~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~ DP:<"dp1"> policy evaluation ~~~~~~~~~~~~~~~
            -->Evaluate Policy: "TemperatureControl"<--
            -->Execute Action: "Start"<--
            -->Evaluate TRC: "ErrorCheck"<--
            -->Execute Action: "Mode3"<--
            -->Evaluate ReturnValue: "IsDecision"<--
DP:<dp1> return value -->0
Kp=9.360000,Ti=5.850000, Td=3.746250, eps=100.000000
w=10.000000, y=0.000000, u=35158.500000
~~~~~~~~~~~~~~ DP:<"dp1"> evaluation ~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~ DP:<"dp1"> context range check ~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~ DP:<"dp1"> policy evaluation ~~~~~~~~~~~~~~~
            -->Evaluate Policy: "TemperatureControl"<--
            -->Execute Action: "Start"<--
            -->Evaluate TRC: "ErrorCheck"<--
            -->Execute Action: "Mode3"<--
            -->Evaluate ReturnValue: "IsDecision"<--
DP:<dp1> return value -->0
Kp=9.360000,Ti=5.850000, Td=3.746250, eps=100.000000
w=10.000000, y=0.000000, u=93.760000
```

Key:

Kp	- the proportional gain		u	- control signal
T_i	- the integration gain		y	- temperature
T_d	- the differentiation gain		w	- reference value
			eps	- control error

Figure 6. Simulation Trace

show that the architecture makes possible to achieve better control quality even if it is working based on a very simple supervision/tuning policy.

The Open Decision Point architecture supports on-line policy change without the need of re-deployment of the whole control component. This means that any supervision/tuning policy can be replaced at any time with a better, more sophisticated version or a version which was designed for a completely different optimization criteria or control goals.

All the presented mechanisms are optimized for resource-constraint systems. As a result they guarantee a very efficient decision-making process whilst the need for processing power (and system memory) remains reasonably small. In this way presented technology can be implemented using cheap embedded systems.

10. Future Work

The important aspect which needs to be addressed is the problem of entire system stability which in the context of the higher layer supervision adding. In practice it may be very difficult to simulate the whole range of possible values that the environment variables can take. Much better way of proving that the complex (policy-supervised) system does not deteriorate stability margins and remains not less predictable than its simpler (fixed) version would be a formal method. However, there is no existing methodology dedicated for formal verification of AGILE policies. A natural solution in this situation is, either:

- Design a formal tools for AGILE policies verification.
- Adapt an existing method of formal verification of dynamic systems. The most promising way seems to be the Petri nets.

Actually, the possibility of using Petri nets formalism for validation of AGILE policies and thus validating the whole policy-supervised system are in progress and will be subject of the ongoing publications.

References

[1] R. Anthony: (n.d.) http://www.policyautonomics.net.

[2] R.J. Anthony: A policy-definition language and prototype implementation library for policy-based autonomic systems. *Proc. of 3rd Int. Conf. on Autonomic Computing (ICAC2006)*, (2006), 265-276.

[3] R.J. Anthony, M. Pelc and W. Byrski: Context-aware reconfiguration of autonomic managers in real-time control applicaitons. *Proc. of 7th IEEE Conf. on Autonomic Computing*, (2010), 73-74.

[4] R.J. Anthony, M. Pelc, P. Ward and J. Hawthorne: Flexible and robust run-time configuration for self-managing systems. *SASO '08: Proc. of the 2008 Second IEEE Int. Conf. on Self-Adaptive and Self-Organizing Systems*, (2008), 491-492.

[5] W. Byrski: Observers and their applications in adaptive control systems. *Scientific Bulletins of The University of Mining and Metallurgy*, **1551**(65), (1993).

[6] W. Byrski and S. Fuksa: Optimal finite parameter observer. An application to synthesis of stabilizing feedback for a linear system. *Control and Cybernetics* **13**(1), (1984).

[7] W. BYRSKI and M. PELC: Modelling and simulation of state observers in the computer control systems. *Proc. of 39th Int. Conf. on Modelling and Simulation of Systems*, (2005).

[8] W. BYRSKI and M. PELC: Continuous and discrete integral state observers in on-line control systems. *Proc. of 39th Int. Conf. on Modelling and Simulation of Systems*, (2006).

[9] W. CHIA-JU: Genetic tuning of PID controllers using a neural network model: A seesaw example. *J. Intell. Robotics Syst.*, **25** (1999), 43-59.

[10] J.I. CORCAU and E. STOENESCU: An adaptive pid fuzzy controller for synchronous generator. World Scientific and Engineering Academy and Society (WSEAS), (2008).

[11] T. FUJINAKA, Y. KISHIDA, M. YOSHIOKA and S. OMATU: Stabilization of double inverted pendulum with self-tuning neuro-PID. *IEEE Computer Society*, (2000).

[12] G. JIE and T. SHENGJING: Application of parameter self-tuning fuzzy PID controller in guidance loop of unmanned aircraft. *IEEE Computer Society*, (2009).

[13] O. KARAKASAL, E. YESIL, M. GÜZELKAYA and I. EKSIN: Implementation of a new self-tuning fuzzy pid controller on PLC. *Turkish J. of Electrical Engineering*, **13**(2), (2005), 277-286.

[14] C. KUAN-YU, T. PI-CHENG, T. MONG-TAO ÅND F. YI-HUA: A self-tuning fuzzy PID-type controller design for unbalance compensation in an active magnetic bearing. *Expert Syst. Appl.*, **36** (2009), 8560-8570.

[15] J.A.M. LIMA and A.E. RUANO: Neuro-genetic pid autotuning: time invariant case. *Math. Comput. Simul.*, **51** (2000), 287-300.

[16] F. LIN, R.D. BRANDT, and G. SAIKALIS: Self-tuning of PID controllers by adaptive interaction. *Proc. of American Control Conf.*, 2000, 3676-1681.

[17] M. PELC, R. ANTHONY and W. BYRSKI: Policy supervised exact state reconstruction in real-time embedded control systems. *Proc. of 7th Workshop on Advanced Control and Diagnostics ACD2009*, (2009).

[18] M. PELC, R. ANTHONY and W. BYRSKI: Context-aware real-time systems with autonomic controllers, *Proc. of 5th Int. Conf. on Pervasive Computing and Applications*, (2010).

[19] H. SHIUH-JER and L. YI-HO: Metal chamber temperature control by using fuzzy pid gain auto-tuning strategy, *WSEAS Trans. Sys. Ctrl.*, **4** (2009), 1-10.

[20] P. WARD, M. PELC, J. HAWTHORNE and R.J. ANTHONY: Embedding dynamic behavior into a self-configuring software system. *Proc. of 5th Int. Conf. on Autonomic and Trusted Computing (Springer LNCS)*, (2008), 373-387.

[21] J. YAU-TARNG, C. YUN-TIEN and H. CHIH-PENG: Design of fuzzy PID controllers using modified triangular membership functions. *Inf. Sci.*, **178** (2008), 1325-1333.

[22] A. YAZDIZADEH, A. MEHRAFROOZ, J. JOUZDANI and R. BARZAMINI: Adaptive neuro-PID controller design with application to nonlinear water level in NEKA power plant. *J. of Applied Sciences*, (2009).

[23] M. ZAHEER-UDDIN and N. TUDOROIU: Neuro-PID tracking control of a discharge air temperature system. *Energy Conversion and Management*, (2004).

Appendix

```
<!-- Policy Definition XML file:
 Policy Language version 1.2 -->
<!-- Application: Temperature Control -->
<PolicySuite PolicyType="PID Controller Tuning">
  <EnvironmentVariables>
    <EVariable Name="ActualError" Type="real"/>
  </EnvironmentVariables>
  <InternalVariables>
    <IVariable Name="ReferenceError" Type="real"/>
  </InternalVariables>
  <OutputVariables>
    <OVariable Name="alpha" Type="real"/>
    <OVariable Name="beta" Type="real"/>
    <OVariable Name="gamma" Type="real"/>
  </OutputVariales>
  <Templates>
    <Template Name="SetParamaters">
      <Assign Variable="ReferenceError"
       Value="5"/>
    </Template>
  </Templates>
  <ReturnValues>
    <ReturnValue Name="IsDecision" Value="0"/>
    <ReturnValue Name="NoDecision" Value="1"/>
  </ReturnValues>
  <Actions>
    <Action Name="Start">
      <EvaluateTRC TRC="ErrorCheck"/>
      <Return ReturnValue="NoDecision"/>
    </Acton>
    <Action Name="Mode1">
      <Assign LHS="alpha" RHS="3.0"/>
      <Assign LHS="beta" RHS="1.0"/>
      <Assign LHS="gamma" RHS="1.125"/>
      <Return ReturnValue="IsDecision"/>
    </Action>
    <Action Name="Mode2">
```

```
      <Assign LHS="alpha" RHS="3.0"/>
      <Assign LHS="beta" RHS="2.0"/>
      <Assign LHS="gamma" RHS="1.125"/>
      <Return ReturnValue="IsDecision"/>
    </Action>
    <Action Name="Mode3">
      <Assign LHS="alpha" RHS="3.0"/>
      <Assign LHS="beta" RHS="3.0"/>
      <Assign LHS="gamma" RHS="1.125"/>
      <Return ReturnValue="IsDecision"/>
    </Action>
  </Actions>
  <ToleranceRangeChecks>
    <TRC Name="ErrorCheck" Check="ReferenceError"
     Compare="ActualError" Tolerance="2"
     ActionInZone="Mode2" ActionHigher="Mode1"
     ActionLower="Mode3"/>
  <ToleranceRangeChecks/>
  <Policies>
    <Policy Name="TemperatureControl"
     PolicyType="NormalPolicy">
      <Load Template="SetParamaters"/>
      <Execute Action="Start"/>
    </Policy>
  </Policies>
</PolicySuite>
```