THE ADAPTIVE PID CONTROLLER USING OPC TOOLBOX

T. Chvostek, M. Foltin, Ľ. Farkas

Syprin s. r. o. , Žehrianska 10, 851 05 Bratislava, Slovak Republic

1 INTRODUCTION

Current automation theory offers us many modern and robust methods of process control which can be developed directly and easily in a Matlab environment. Control engineers and students can immediately apply various theories, build complex models and simulate them on standard PCs without any difficulties. On the other side, deploying control system quickly from "simulink" to real system can be sometimes quite complicated task. Usually additional hardware (to process input/output values) and corresponding software (to realize control algorithm) is required to accomplish this job. It is commonly implemented on proprietary hardware with some proprietary software which simplifies implementation and integration process. However this solution decreases transparency (software is not open-source in most cases), usability/portability (only components from one manufacturer are usable) and also raises costs of whole system. This paper attempts to show you how to quickly build PLC – PC based adaptive control system with implementation of OPC technology.

Nowadays programmable logic controllers (PLCs) are wide-spread and used in many industrial applications (for ex. automobile industry). PLC as an independent unit can contain complex programs and operate at high speeds with hundredths of inputs/outputs. We only have to follow several rules of ladder programming and we can realize almost any control program. Sometimes it is very difficult to realize complex algorithms in a PLC, especially when we need large amount of memory or complicated data manipulation. To avoid this situation we can separate this problem to two parts. In the first part we will use PLC as a real-time controller, but only with simple program and support for I/O operations. Next part will be realized on a standard PC with user defined application (e.g Matlab). This part of control software can contain complex algorithms and data. Both parts will be connected through OPC (Open Process Control) communication channel which allow them to exchange data. OPC communication is provided by OPC server software which runs on a PC connected to PLC. Running application acts as OPC client and can sends computed data to OPC server. OPC server accepts data from OPC clients (through OPC protocol) and delivers them directly to a PLC. A client sends requests to OPC server and receives values of PLC internal variables in a similar way they also set PLC variables. OPC communication uses OPC protocol which is defined as standard and is independent on a used PLC. In this paper we used Matlab OPC client implementation.

2 CONTROL SYSTEM WITH OPC COMMUNICATION

Open process control (OPC) is a series of seven specifications defined by the OPC Foundation (http://www.opcfoundation.org) for supporting open connectivity in industrial automation. OPC uses Microsoft DCOM technology to provide a communication link between OPC servers and OPC clients. OPC has been designed to provide reliable communication of information in a process plant, such as a petrochemical refinery or an automobile assembly line.

2.1 Control system overview

Our control system is separated into two parts. First part is called process control layer and contains process - small DC Motor and a controller - PLC SM-200 (Siemens). The second part contains standard PC and our control software in Matlab. In this paper we used PLC only as a real-time PID (PSD) controller. The control PC was standard office PC connected to PLC with preinstalled

Matlab 7.0.4. The Matlab environment offers us smart set of OPC client functions so we built adaptive control application in a Matlab (with Simulink). The control application (OPC client) can be applied on any PC which has network connection to PC with OPC server, but we used only one PC solution (both OPC server and OPC client running on the same machine). The whole system is on figure 1.

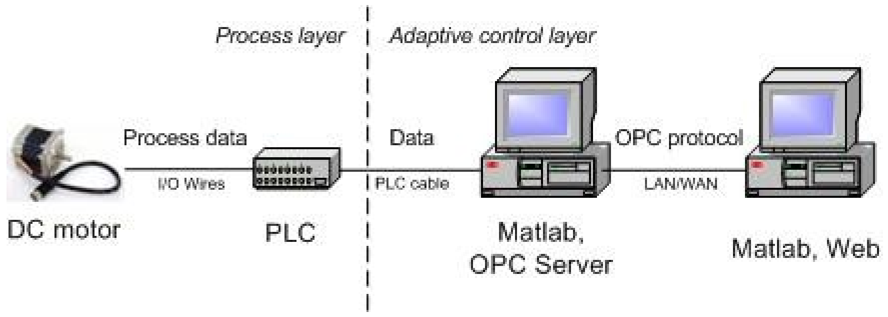


Figure 1 – Control system with OPC communication

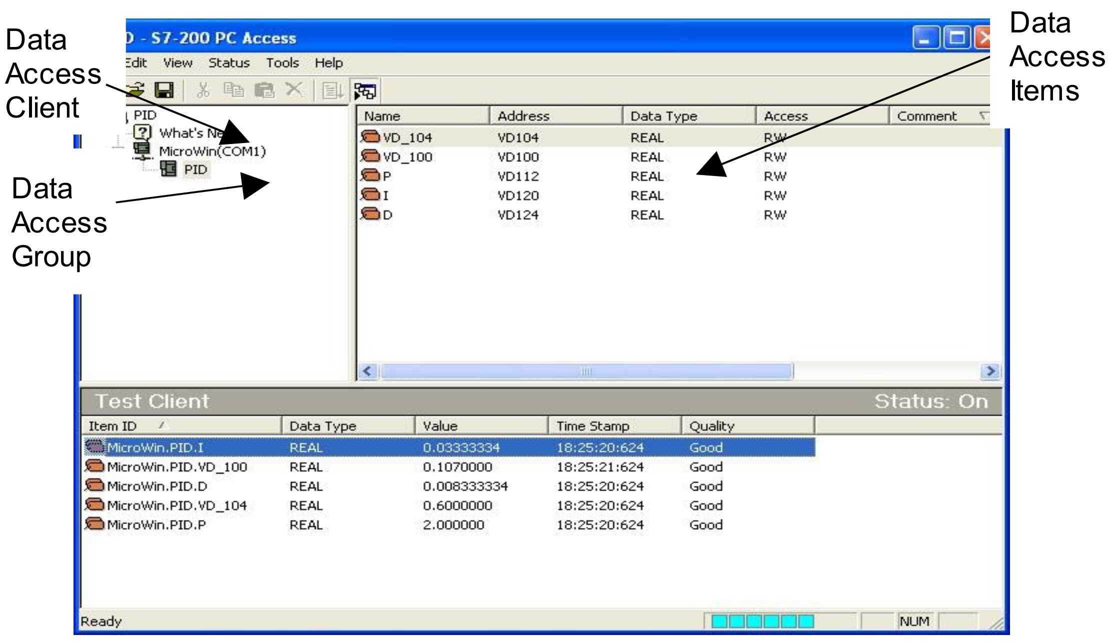
2.2 OPC communication principle

In this part of document we sketch some basics OPC communication principles. At the beginning consider a situation that you want to read information from PLC. Firstly you have to join your PC to PLC with appropriate cable. In next step you have to install OPC server software (contact your PLC manufacturer) and configure it (server name, etc.). In the last step you have to define which variables you want to read from PLC and make available as OPC variables to OPC clients (e.g. your application). Variables can be defined manually in OPC server or at runtime from OPC client. We used second solution because it is more practical. Definition of OPC variable is simple binding between PLC variable (for ex. M0.0, VD100, I0.1, …) and variable name (which defined by user).

The OPC Toolbox in Matlab is implemented using three basic objects, designed to help us manage connections to servers and collections of server items.

OPC Toolbox Object Hierarchy:

- OPC Data Access Client objects
  represents a specific OPC client instance that can communicate with only one server. You define the server using the Host and ServerID properties. The OPC Data Access client object acts as a container for multiple group objects, and manages the connection to the server, communication with the server, and server name space browsing.
- Data Access Group objects
  represents containers for one or more server items (data points on the server.) It manages how often the items in the group must be read, whether historical item information must be stored, and also manages creation and deletion of items.
- Data Access Item objects
  represents server items. Items are defined by an item ID, which uniquely defines that server item in the server's name space. A DAItem object has a Value, a Quality, and a TimeStamp, representing the information collected by the server from an instrument or data point in a SCADA system.



Name	Address	Data Type	Access	Comment
VD_104	VD104	REAL	RW	
VD_100	VD100	REAL	RW	
P	VD112	REAL	RW	
I	VD120	REAL	RW	
D	VD124	REAL	RW	

Item ID	Data Type	Value	Time Stamp	Quality
MicroWin.PID.I	REAL	0.03333334	18:25:20:624	Good
MicroWin.PID.VD_100	REAL	0.1070000	18:25:21:624	Good
MicroWin.PID.D	REAL	0.008333334	18:25:20:624	Good
MicroWin.PID.VD_104	REAL	0.6000000	18:25:20:624	Good
MicroWin.PID.P	REAL	2.000000	18:25:20:624	Good

Figure 2 – OPC server main window

Now we demonstrate how to define set of OPC variables from Matlab m-file on a simple practical example:

Firstly we connect to our OPC server (named S7200.OPCServer) and add new group of OPC variables:

```
        da = opcda('localhost', 'S7200.OPCServer'); % OPC Data Access Client Object
        connect(da);
        grp = addgroup(da); % OPC Data Access Group object
```

Text localhost can be replaced with valid IP address of our OPC server (but we have OPC server and client on the same machine, so localhost is sufficient)

Next we add some process variables:

```
% PID values define in PID block of PLC
itmP = additem(grp, 'MicroWin.PID.P'); % OPC Data item object
itmI = additem(grp, 'MicroWin.PID.I');
itmD = additem(grp, 'MicroWin.PID.D');

itmHod = additem(grp, 'MicroWin:2:0.0.0.0:0000:0000,VD108,REAL,RW,0.0000000,0.0000000');
itmTab = additem(grp, 'MicroWin:2:0.0.0.0:0000:0000,M0.0,BOOL,RW,0.0000000,0.0000000');
% array of OPC variables

for i=0:1:99
        d = 200 + 4*i;
        string = sprintf('MicroWin:2:0.0.0.0:0000:0000,VD%d,REAL,RW,0.0000000,0.0000000', d);
        itmH(i+1) = additem(grp, string);
end
```

If we now want to write new values to PID we can use commands:

```
result=writeasync(itmP, PP);
result=writeasync(itmI, Ti);
result=writeasync(itmD, Td);
```

If we want to immediately read values of PID controller:

```
result=readasync(grp, ‘'MicroWin.PID.P’);
result=readasync(grp, ‘'MicroWin.PID.I’);
result=readasync(grp, ‘'MicroWin.PID.D’);
```

If we want to read all variables in a group, we use command:

```
r=read(grp);
```

we can get OPC variable value with commad:

```
h = itmH(3).Value
tab = itmTab.Value;
p = itmP.Value
```

3 ADAPTIVE HEURISTIC PID CONTROLLER

For the sake of simplicity we will describe only fundamental methods and features of our adaptive heuristic controller. As a control algorithm we implemented adaptive heuristic PID controller based on process output recognition. Detailed description is in [Foltin M., 2000].

3.1 Preparations

Transfer function of controlled laboratory DC motor was (motor load was set to 3V):

$$G(s) = \frac{107}{11s^3 + 11s^2 + 91s + 108.3} \tag{1}$$

Operating point:

$$U_0 = 6.7V \quad Y_0 = 5V \tag{2}$$

Step response of DC motor was:

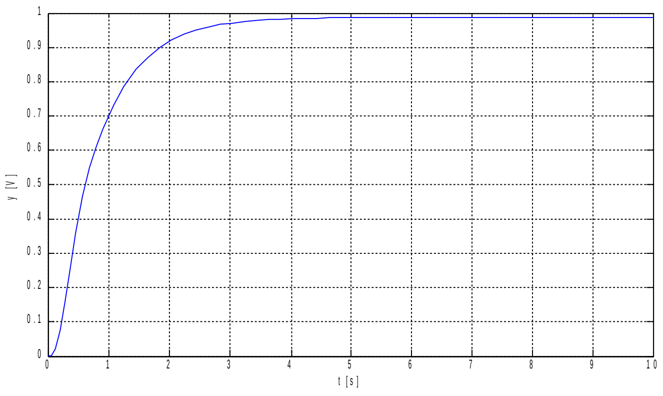


Figure 3 – DC motor model step response

We designed an optimal PID controller for DC motor. We request fast control and no overshoot. Our sampling period was set to 100 milliseconds and process output was restricted to <0,10V>. Designed PID will be called PID* which stands for the best PID.

PID* coefficients of our motor controller were:

$$P^* = 1.4, I^* = 1, D^* = 0.8 \tag{3}$$

Closed loop response with controller:

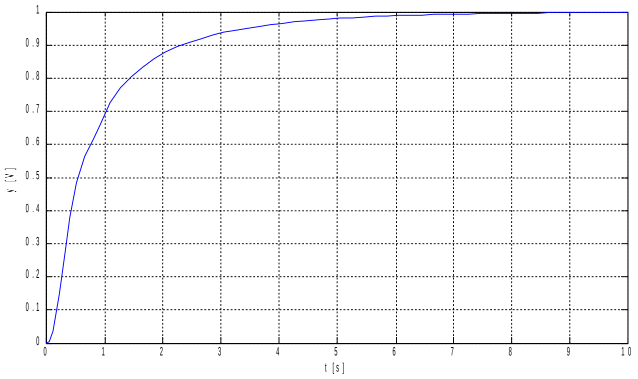


Figure 4 – Closed loop response of DC motor with PID*

In next step we defined 3D mesh of PID coefficients ranges for base rule generation:

$$\begin{aligned} \langle \Delta P_{\min}, \Delta P_{\max}, dP \rangle &= \langle -0.6, 0.6, 0.2 \rangle \\ \langle \Delta I_{\min}, \Delta I_{\max}, dI \rangle &= \langle -0.6, 0.6, 0.2 \rangle \\ \langle \Delta D_{\min}, \Delta D_{\max}, dD \rangle &= \langle -0.3, 0.3, 0.1 \rangle \end{aligned} \tag{4}$$

We use this as input for our rule base generator which generated step responses for every point of PID 3D mesh (Fig. 5).

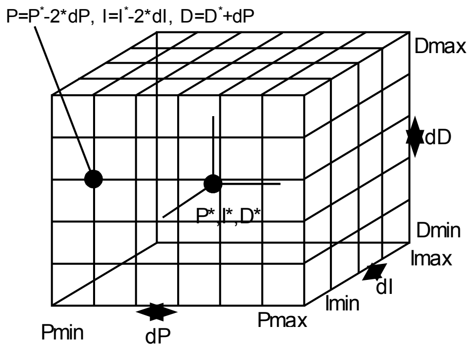


Figure 5 – 3D mesh of PID coefficients

This prepared database of control rules (pair: deviation of PID from PID* and corresponding closed loop response) was later used for process control in our software.

3.2 Algorithm basic principle

The method assumes control structure as shown in Fig. 6. We can shortly describe control algorithm method as:

1. waiting for chosen kind of error (parameters changed, noise, reference signal changed…)
2. evaluation of chosen process parameters after error from first point
3. compute new differences of PID coefficients from the best PID (according to base of rules)
4. correction of controller coefficients
5. jump back to 1.

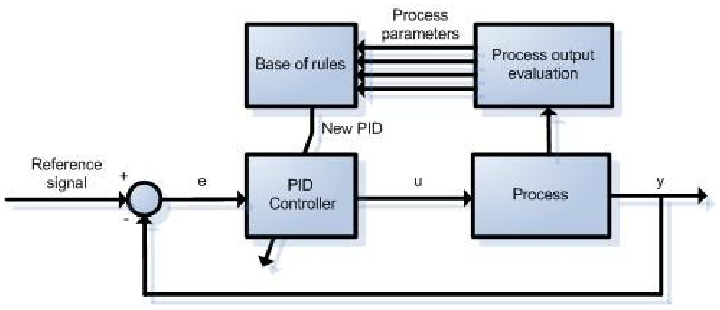


Figure 6 –Basic principle of adaptive controller

In a more detail we start algorithm with some randomly chosen PID. The main part of algorithm measure and compare differences between database rules and current process output values after each change of reference signal. Before next change of reference signal we choose rule with the lowest least square difference and according to this rule we update values of PID controller in the next step of reference signal. Algorithm should converge to desired output (Fig. 4). Reference signal was generated as a series of steps from 1 to 0V around operating point (2).

4 PRACTICAL EXPERIMENTS

We used Matlab for adaptation of PID coefficients and PLC as standard PID controller. Process output values needed for adaptation is acquired from OPC server. After whole adaptation process we set new values of PID coefficients to PLC. In experiments we set the initial PID (PID is in software automatically converted to PSD) to some values which are outside the defined base of rules and start the algorithm (PID_1, PID_2, PID_3). We can see results from Fig. 7 to Fig. 9.

4.1 Experiments

PID_1 (5) is outside designed base of rules. Reference signal is series of steps from one to zero around operating point. Process is controlled by PSD controller in a PLC, after each change of reference signal new PSD values (optimal) are taken from heuristic algorithm and sent to PLC. Process output reaches desired output. Results are on Fig. 7.

$$P_1 = 1, I_1 = 1.9, D_1 = 0.5 \tag{5}$$

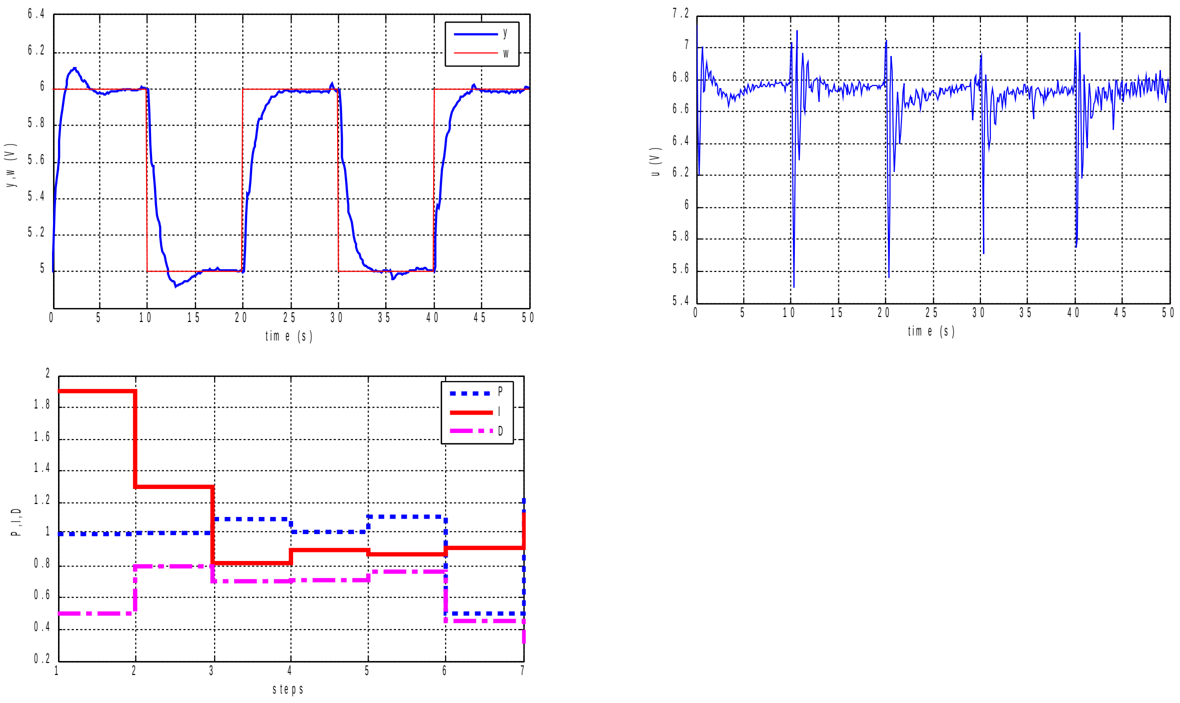


Figure 7 – Adaptive heuristic controller started with start PID=PID1

Fig. 8 shows another adaptation process (with start PID=PID$_2$ (6)) and also gives stable results. We can also reverse this technique and make parameter changes in a controlled process. These results are shown on Fig. 9.

$$P_2 = 0.7, I_2 = 2.8, D_2 = 0.5 \qquad (6)$$

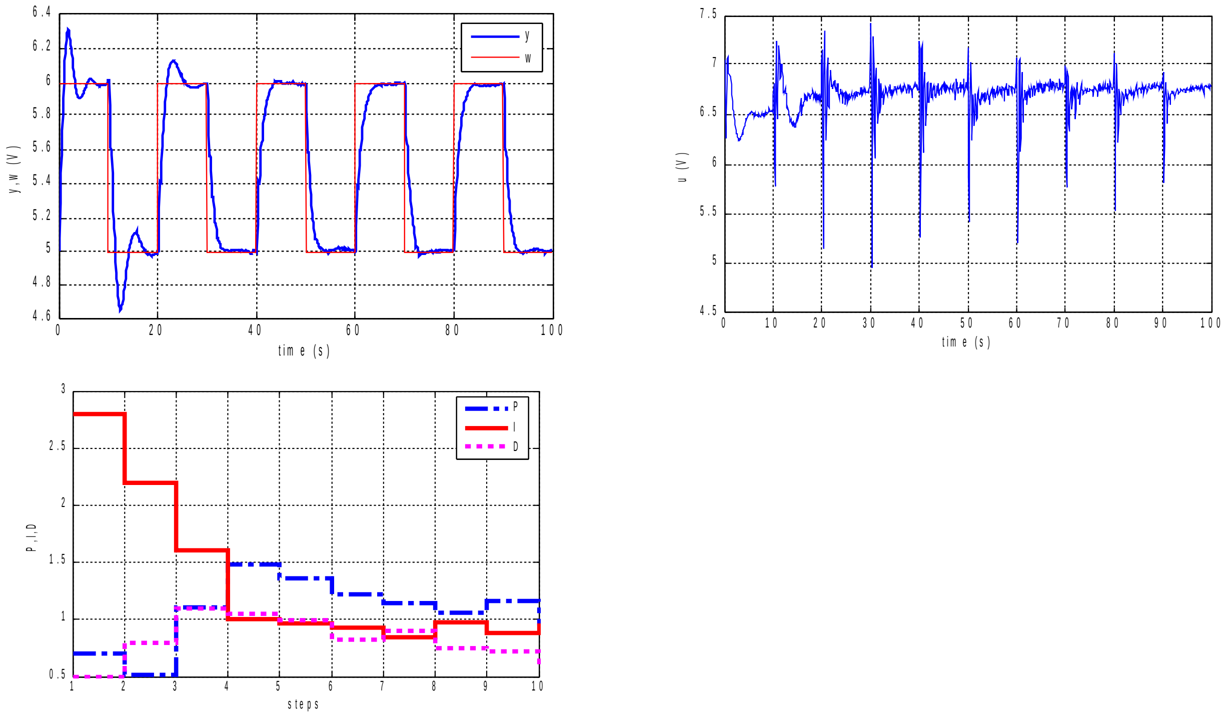


Figure 8 – Adaptive heuristic controller start PID=PID2

$$P_3 = 1, I_3 = 2.3, D_3 = 1.1 \qquad (7)$$

In the 3rd test we set start PID to PID_3 (7).We start algorithm and make several steps to get desired process output. In step eight of reference signal (t=80s) we made step with motor load from 3V to 4.5V (Fig. 9). Adaptive controller stabilized our process.

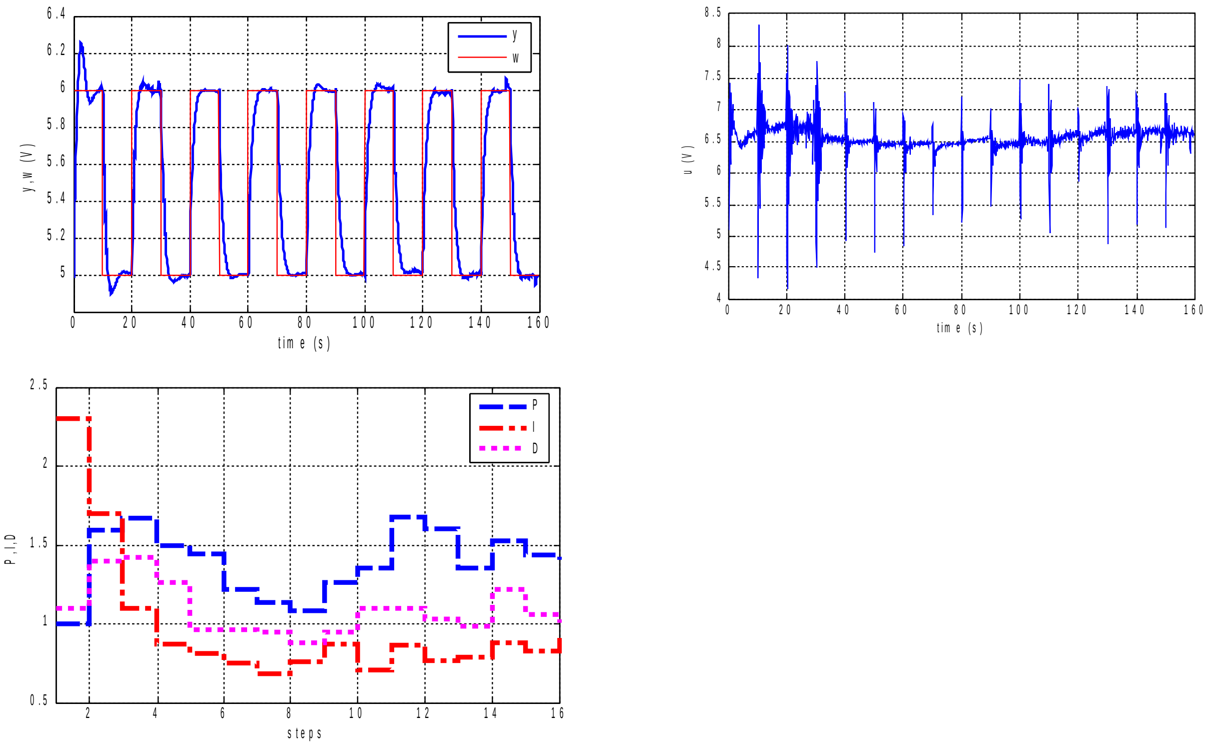


Figure 9 – Adaptive heuristic controller start PID=PID_3, motor load step at t=80s from 3V to 4.5V.

6 SUMMARY

We approved that OPC technology together with Matlab can be used for adaptive control of DC motor. Practical experiments also prove that adaptive algorithm achieves requested control results and is stable for a wide range of process parameters. OPC technology as a communication interface with standard PLC provides us modern approach to process control. Almost every PLC on the market has OPC server software which acts as communication gate for other applications to PLC. Third party applications can realize VRML visualization, database archiving, remote access to PLC data or remote control and many more. Benefits from OPC control are that we can monitor or exchange any data with PLC (base upon OPC server settings) through Intranet/Internet network. We also don‘t need any additional I/O card (PLC reads and sets inputs and outputs) and because we use PLC at low level layer, we achieve true real-time control. This solution is also PC crash proof. If a control PC crashes, PLC still continues in real-time control (PC crash results only control quality degradation). OPC control can be also (with help of web technologies) easily integrated into web control.

ACKNOWLEDGEMENTS

This work has been supported by the Slovak Research and Development Agency, Grant No. APVV-99-045805.

7 REFERENCES

[1] CHVOSTEK, T.: 2003, *Master of Science thesis – Adaptívne riadenie v distribuovanom riadiacom systéme*, KASR FEI STU Bratislava.

[2] DLUHÝ, M.: 2004, *Master of Science thesis*, KASR FEI STU Bratislava, 2004.

[3] FOLTIN, M.: 2000, *Master of Science thesis – Znalostne založené adaptívne riadenie*, KASR FEI STU Bratislava.

[4] MARŠÍK, J.: *Algoritmy pro jednoduché adaptivní číslicové regulátory PSD*, Zborník prednášok za 7. celoštátnej konferencie ASRTP 86 Tále – Nízke Tatry, 1986

[5] MATLAB OPC toolbox help, Dostupné na http://www.mathworks.com

[6] OPC FOUNDATION, *Specification of OPC*, Dostupné na : http://www.opcfoundation.org

[7] PFEIFFER, B. M.; ISERMANN, R.: Selftuning *of classical controllers with fuzzy-logic*, Proc. of IMACS Symp. "Mathematical and Intelligent models in system Simulation", Brussels 1993.

[8] DESILVA, C. W.: *An Analytical Framework for Knowledge-Based Tuning of Servo Controllers*, Eng. Applic. Artif. Intellig., Vol. 4, 1991, No. 3, 177-189.

[9] SEKAJ, I.; FOLTIN, M.: *Adaptive Control Based on the Closed-Loop Response Recognition*. VDI Berichte, 2000, Nr. 1761-1805, 82-88.

CONTACTS

www.syprin.sk

tomas.chvostek@syprin.sk

martin.foltin@syprin.sk

ludovit.farkas@syprin.sk