A New PC-Based Workbench for Virtual Instrumentation and Automatic Control Using Matlab GUI/MEX-C++ Application

JEAN MBIHI
EEAT (Electrical, Electronic, Automation and Telecommunications) Research Laboratory
Advanced Teachers' Training College for Technical Education
University of Douala, BP 1872, Douala, Cameroon
mbibidr@yahoo.fr , http://www.cyberquebec.ca/mbihi/

Abstract: - This paper presents an original PC-Based Workbench for virtual instrumentation and automatic control. Its software platform results from a mix of Matlab *Graphical User Interface* (GUI) design and *Matlab Executable* (MEX) C++ programming. It is shown how a sophisticated custom MEX C++ control library, callable from Matlab GUI application, has been built from the C++ driver of an arbitrary data acquisition board. The varieties of simulation and experimentation modes provided include: digital-to-digital controls, digital-to-analog and analog-to-digital conversion, open loop control, and Matlab GUI/MEX controllers. A sample of simulated and experimental results obtained and presented, shows the great merit of the proposed well tested PC-based workbench in control engineering education.

Key-Words: - PC-Based Workbench, Virtual instrumentation, Automatic control, Matlab GUI, MEX-C++, mexFunction, data acquisition board, Advanced Programming, Matlab/MEX controllers.

1 Introduction

In engineering education, workbenches are widely used as didactic equipments in laboratories and workshops. They enable learners to study the experimental behavior of a variety of real systems used or to be used in the professional or social world. The experimentation is an efficient means to reinforce the understanding of conceptual models and properties, learned when theoretical lessons are taught in classrooms or amphitheaters. In addition, the experiments conducted on workbenches also enable students to acquire practical knowledge and professional skills about the use of real systems.

Generally speaking, the demand of workbenches for educational purposes takes into account a number of requirements including: size, ease of use, flexibility, interactivity, experiments options, monitoring comfort, communication performance, ease of extension, robustness, security and costs. Hence, the need of building powerful solutions to these multiples requirements, have motivated over years a rapid development of virtual workbenches with local or remote access [1]-[2]. At the heart of a virtual workbench is a digital computer, i.e., a PLC (programmable logic controller), an industrial PC (personnel computer) or a standard PC. A sample of digital computers types used nowadays for building virtual workbenches is presented in Fig. 1. In each case, there are specific constraints about the additional peripherals, the digital control platform and the maintainability.

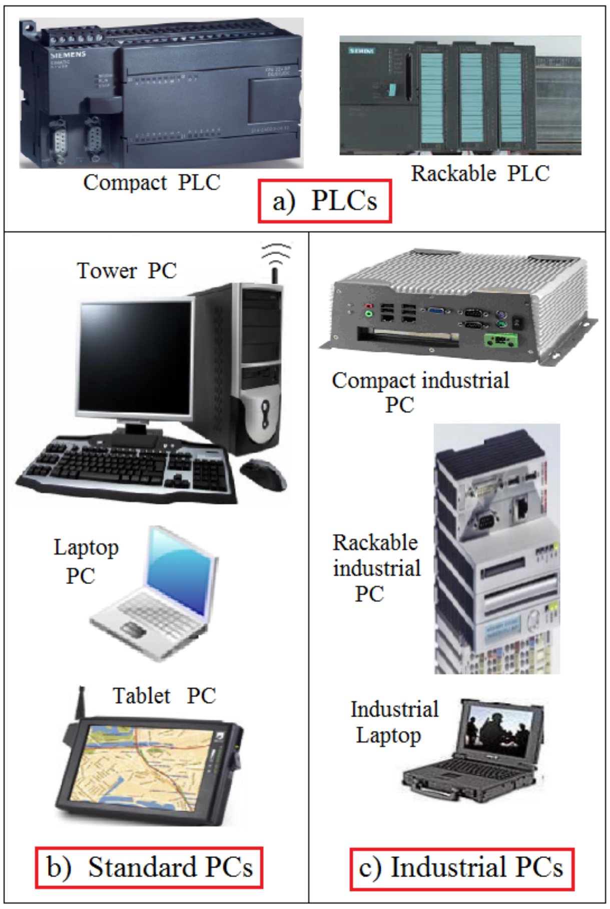


Fig. 1 Sample of digital computers.

A long time ago, because of robustness and low cost requirements, PLCs were the most efficient equipments for instrumentation and automatic control in industry. However, compared to PCs, their use today at the heart of virtual workbenches in engineering education is far to be a merit since :

a) PLC programming needs the availability of a complete PC for developing and compiling the automation software, and uploading the resulting objet code into embedded memory;
b) PLC programming languages lack building resources for sophisticated control policies, e.g, feedback control with state observers, self tuning control, intelligent control from fuzzy logic or neural network models of dynamic plants, optimal or predictive control, stochastic control with/without Kalman state estimator.
c) PLC programming framework lacks sufficient capacity memory for data monitoring or logging over a long time period.
d) PLC-based virtual instrumentation needs either a very costly visualization equipment, e.g., a Magelis, or a supervision PC driven by a development tool with drivers for PLC targets such as Labview, Cadepa or Automgen.
e) PLC operating systems very have limited capabilities in terms of digital processing speed of tasks, compared to Windows or Linux platforms used for PC-based controls.

More details about weaknesses of PLCs compared to PCs when building instrumentation and control systems, are outlined in [3]. Following these comparatives studies, it is clear that the overall cost of a PLC-based workbench appears to be more significant than that of a PC-based workbench with identical performance and quality. At this point, it is important to mention that industrial PCs are as robust as PLCs (see Fig. 1). In addition, they support the same type of operating system, development software, communication bus, hardware, digital input-output ports, data acquisition boards and external peripherals than the standard PCs. However, for the same characteristics and performance, industrial PCs are more costly than standard PCs. For these main raisons, standard PCs appears to be the better building technology for virtual workbenches in engineering education.

PC-Based workbenches encountered in modern engineering, consist of power processes, hardware interfaces, and software control applications [4]-[9] In control engineering education, Matlab is increasingly used as the most versatile development tool, for rapid creating virtual instruments and controllers with advanced GUI (graphical user interface) ([7], [8], [10]). Nowadays this growing popularity is reinforced by *data acquisition* (DAQ) toolboxes available in recent Matlab editions [11].

However, the real time feasibility of virtual instrumentation and control under Matlab is dictated by the availability and capabilities of DAQ (data acquisition) drivers in Matlab/Simulink DAQ Toolboxes [11]. Unfortunately:

a) Recent Matlab editions do not support all types of DAQs available in the market;
b) DAQ drivers supported by recent Matlab editions only provide basic input-output functions, at the expend of instrumentation resources and real time control policies;
c) Real time control policies can be only implemented as Matlab/Simulink blocks;
d) A digital feedback controller consisting of Matlab/Simulink blocks, might become greedy within a fast real time control loop.
e) Updating DAQ board drivers registered in Matlab cannot be done independently by the end user.

In order to overcome these weaknesses, this paper presents a new PC-based workbench for *virtual* instrumentation and automatic control. The novelty of the proposed workbench arises from many facts:

a) Its software platform relies on Matlab GUI and MEX (Matlab Executable) C++;
b) Its precompiled MEX-C++ library provides advanced functions and real time controllers;
c) A custom *DAQ* board is supported even though it is not registered in Matlab;
d) A wide variety of experiments are provided.

The prototyping realizations of the proposed virtual workbench, are in use at the Advanced Teachers' Training College for Technical Education of the university of Douala. Most are used in the electrical and electronic engineering department, in order to reinforce practical skills in instrumentation and control courses, to a mean population of 90 undergraduates per semester. In addition, the GUI/MEX-C++ codes are used as realistic case studies in Master degree II program, when teaching the course entitled *Advanced programming of Automated systems*, to a mean population of 20 graduate students per year.

In Section 2, the hardware architecture of the proposed workbench is presented. Then, the software architecture of Matlab GUI/MEX-C++ application is outlined in Section 3. In Section 4, a custom MEX-C++ library is outlined, followed in

Section 5 by the presentation of experimentation modes and related experimental results obtained when testing the proposed PC-based virtual workbench. Finally, the conclusion of the paper is outlined in Section 6.

2 PC-based workbench Architecture

As shown in Fig. 2, the hardware architecture of the PC-based workbench consists of 3 main subsystems:

- A PC with preinstalled Matlab GUI/MEX-C++ application described later (Fig. 1(a)).
- An USB hardware interface (Fig. 2(b)). It is a low cost Velleman K8055/VM110 DAQ with mean conversion rate 50 Samples/s. It consists of two 8-bits ADC (analog-to-digital conversion) channels with adjustable gains, two 8-bits digital-to-analog conversion (DAC) channels with light indicators, 8 digital-to-digital conversion (DDC) outputs with light indicators and 5 DDC inputs with test buttons. In addition, it is sold with a free drivers for standard programming languages including C++. Although the K8055 DAQ toolkit for Matlab might be already available, It is not necessary in our context because of the aforementioned weaknesses about most Matlab DAQs drivers.
- A power servo system shown in Fig. 2(c), with a load disturbance (magnetic brake).

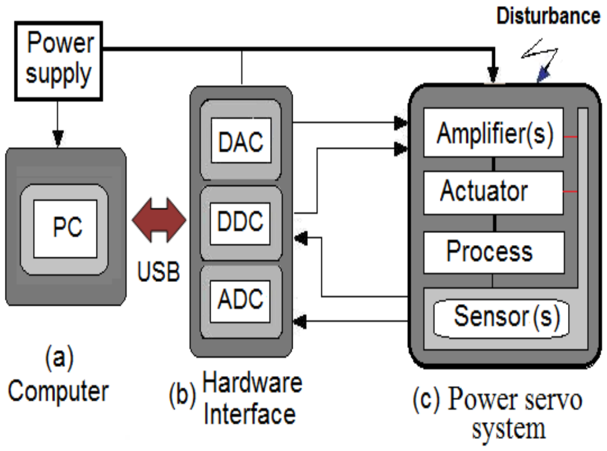
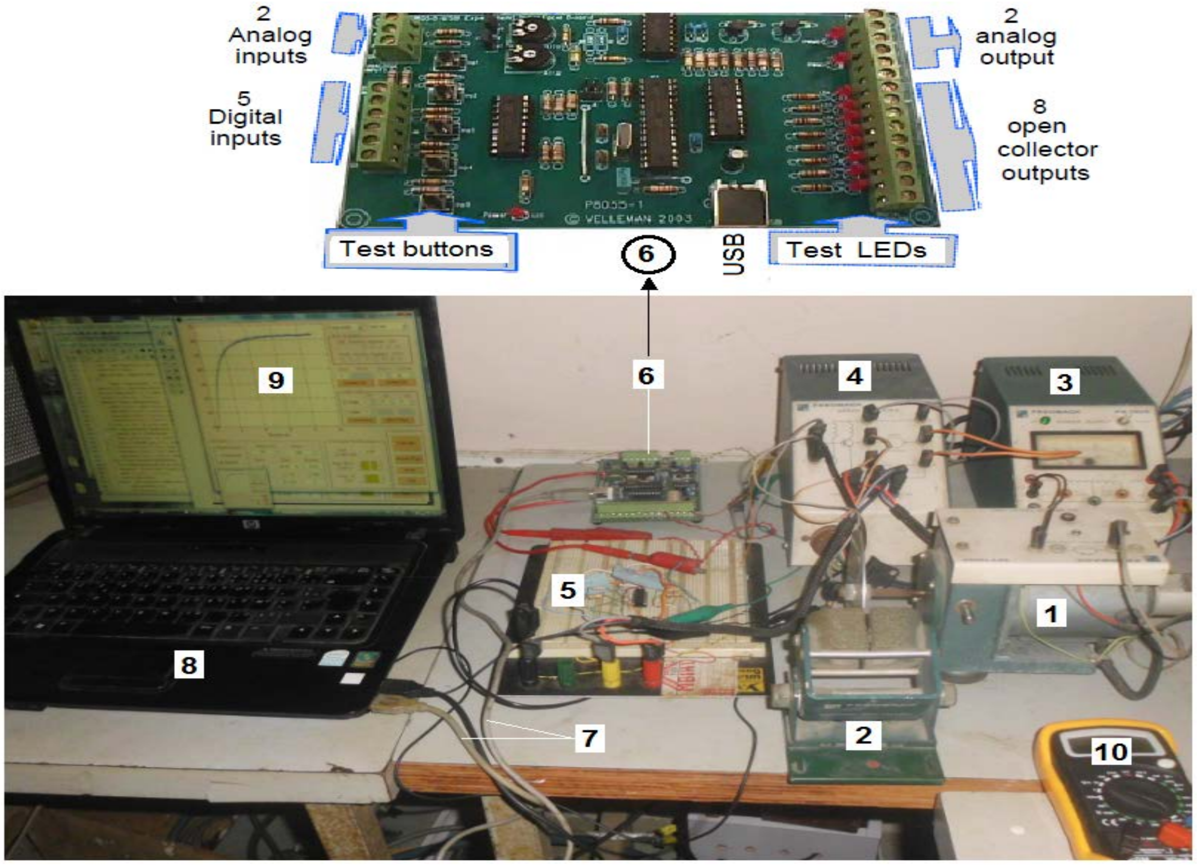


Fig. 2 Synoptic diagram of the PC-based workbench.

A real view of the prototyping PC-based workbench is presented in Fig. 3, where subsystems are identified and numbered from 1 to 10.



1) Servo DC motor -Tacho unit; 2) Magnetic brake; 3) Power supply; 4) Servo Amplifier; 5) Preamplifier; 6) K8055/VM110/USB-MDAQ; 7) USB Cable; 8) PC; 9) GUI/MEX-C++ application; 10) Digital voltmeter

Fig. 3: Hardware architecture of the virtual workbench for a servo system

3 Matlab GUI/MEX-C++ application software

The structure of Matlab GUI/MEX-C++ application software presented in Fig. 4 consists of:

- A visual control panel (Fig. 4(a) with interactive and static visual controls.
- A set of GUI and a Matlab files edited as objects Figure *.Fig and Matlab file *.m respectively as shown in Fig. 4(b). While the file *.Fig contains references of visual objects embedded on the front panel, Matlab file (*.m) contains Matlab Callback functions (Fig. 4(d)), including MEX functions accessible from the GUI as shown.
- A MEX Library in Fig. 4(e)). It consists of MEX functions compiled offline.
- A C++ driver given by the manufacturer of the target DAQ board.

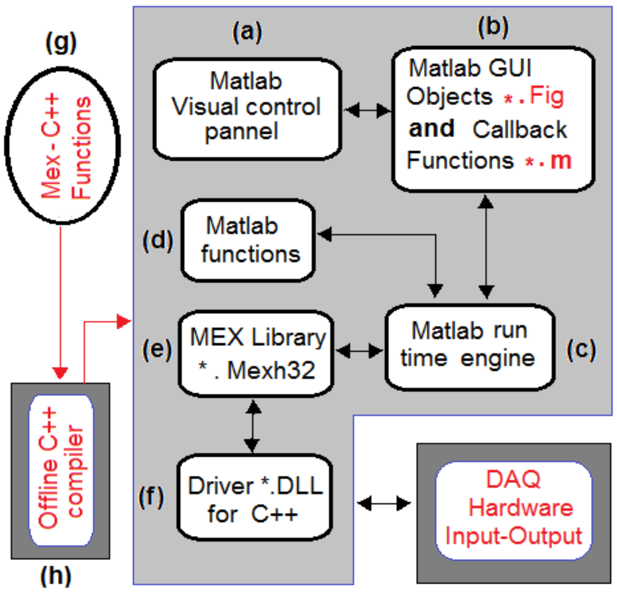


Fig. 4 Structure of Matlab GUI/MEX-C++ application

The calling syntax of a MEX function as presented in Fig. 5, involves *m* output and *n* input variables, each of which being thought off as *mxArray* object.

Furthermore, the structure of MEX C/C++ function as shown in Fig. 6, is similar to a special C/C++ program named *MyMexCpp.C* or *MyMexCpp.Cpp*, and consists of 4 important sections:

a) Standard C/C++ libraries to be used;
b) Special libraries: *<windows.h>*, *"mex.h"*;
c) Global C/C++ structures or functions, and a C++ driver procedure *Myddl*( ) to be called at run-time for loading K8055.DLL driver,

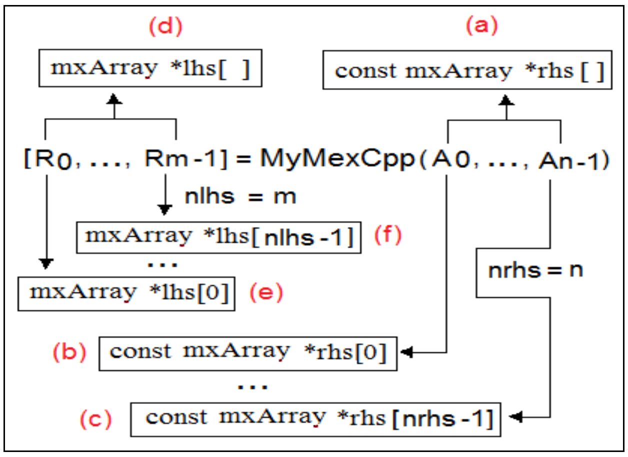


Fig. 5 : Calling syntax of a MEX function available in the proposed MEX-C++ Library.

```
// Mex-C++ Source Program - MyMexCpp.Cpp
// Section I :   Declare   standard C/C++  Library
      #  include  <stdio.h>                // Example 1
      #  include  <math.h>                // Example 2
            ...
      #  include  <math.h>                // Example n
// Section II :  Declare  dedicated  C/C++ Library
       #  include  <windows.h>         // Example 3
       #  include  ''mex.h''              // Example 4
//  Section III :  Define : a)  New global  types of
//         pointers and User data structure  for access
//         et run time to  K8055.DLL  functions ;
//        b) Define  instance Object  for  K8055.DLL
 int  MyDll( )  //  a)  Load  K8055.DLL into RAM
                      //  b)  Instantiate  C++ User Names
                      //       function using  Lib names red
                      //  c)   Open USB-DAQ connection
// Section IV :   Edit the  main entry function
void  mexFunction
       ( int   nlhs,               // Number of  LHS  Args
         mxArray   *plhs[ ],  //  LHS Array pointer
                                    //   from    prhs[0]
         int  nrhs,                // Number of  LHS Args
         const   mxArray  *prhs[ ]    //   Number  of
                                           //   RHS  Args
          )
 {
   // 1)   Declare local variables and   pointeurs
   //        for data Array to be  Imported/Exported
   // 2)   Test  of  imported  Data  *prhs[ ] for errors.
   // 3)   Processing imported data using global
   //        and  local functions, including  MyDll( )
   // 4)   Call  DAQ  output(s)  into  DLL if  needed
   // 5)   Create output Memory for  Array pointer
   //        to be returned to Matlab
   // 6)   Compute  Array output to be returned
   // 7)   Return Array output to Matlab program
  }
```

Fig. 6 Structure of a MEX-C++ function

and initializing connections, for access to hardware ports of the target DAQ board.

d) Main entry function entitled *mexFunction.*

The arguments of a *mexFunction* are:

- ✓ *nlhs:* Number of left hand side outputs
- ✓ **lhs*: Left hand side *mxArray* pointer
- ✓ *nrhs*: Number of right hand side inputs
- ✓ **rhs*: Right hand side *mxArray* pointer

4 MEX-C++ Library for K8055/VM110/USB-DAQ board and access from Matlab GUI

Following MEX programming techniques presented in Figs. 5 and 6, Table 1 summarizes a set of *input-output* (I/O) functions integrated in the proposed MEX-C++ library, built from the basic *K8055.DLL* driver for C++ of the K8055/VM110-DAQ board.

Table 1: Basic K8055.DLL driver f or C++ and the proposed MEX-C++ library

K8055.DLL for C++ and I/O functions	**Proposed MEX- C++ Library and novel I/O functions**
OpenDevice	State = ***open_usb_card*** (0)
Closedevice	***close_usb_card*** ( )
SetDigitalChannel ClearDigitalChannel SetAllDigital ClearAllDigital WriteAllDigital	***ddcout*** (*ddCh*, *Val*) ddCh = 1 to 8 → Bit Number = 9 → Output word = 10 → All output bits Val = 0 – 254 if *ddCh* = 9 = implicit (otherwise).
ReadDigitalChannel ReadAllDigitalChannel	Val = ***ddcinp*** (*ddCh*) *ddCh* = 1-8 for single bit select = 9 for reading a word
ReadAnalogChannel ReadAllAnalog	N = ***adcoutcode*** (*Ch_Dac*) N : 0-253
OutputAnalogChannel OutputAllAnalog SetAnalogChannel SetAllAnalog	***dacinpcode*** (*dac_Ch*, *N1*, *N2*) *dac_Ch* = 1, 2 or 3 (for 1 & 2) *N1*, *N2* : 0 - 253 ;
▨	Val = ***ddc*** (*dir*, *Ch*, *State*) *dir* = 0 (output), 1 (input) *Ch* = 1-8 (bit), 9 (word), 10 (All) *State* = 0 or 1 if *dir* =0 = implicit (otherwise)
▨	***dacout*** (*dac_Ch*, *U*) *dac_Ch* = 1, 2 and 0 ≤ *U* ≤ 4.7 V

▨	U = ***adcinp*** (adc_Ch)
	adc_Ch = 1, 2 or 3 (for 1 & 2);
	0 ≤ U ≤ 4.7 V

▨	Y=**oploco**(*dacCh*,*Uc*,*adcCh*, *Nms*)
▨	Uk_1 = ***UpidMat*** (*Args*)
▨	Uk_1 = ***UpidMex*** (*Args*)

It is a great merit to observe that, unlike most DAQ drivers for Matlab/simulink as in [12], the proposed MEX-C++ library, provides under an arbitrary DAQ with available basic C++ driver, many powerful real time control functions such as *ddc*, *oploco* (open loop control), and UpidMex.

In addition, following Fig. 4, each MEX-C++ function provided in Table 1 for real time instrumentation or control, with extension **.Mexh32* in the current folder, exactly behaves as any standard Matlab function. Thus, it could be called from the callback procedure associated with visual objects available on Matlab GUI used as an advanced virtual instrumentation and control tool.

The general structure of the callback procedure associated with Matlab GUI visual object named *ObjName*, consists of 3 sections as shown in Fig. 7.

```
function    ObjName1_Callback(hObject,  …
                                eventdata,   handles)
%  SECTION I
%      Get Input data from GUI if any
%      for preliminary processing.
%                      Example:
            Val = get(handles.CheckCna, 'Value');
%  SECTION II
%      Process  data  using  Matlab  commands
%      including  MEX  functions
%                      Example:
 if   Val  = = 1
     N1Can  = adcoutcode(1);  % MEX  call
set(handles.TextCanN1,'String',  num2str(N1Can));
     N2Can = adcoutcode (2);
set(handles.TextCanN2, 'String', num2str(N2Can));

 end   %  for  If  command
%  SECTION III
%     Additional  decision making  and  actions
end   %  for function
```

Fig. 7 Structure of the callback function for a GUI visual object.

In Fig. 7, *get* and *set* are Matlab GUI commands allowing to capture and modify respectively, the property of any GUI visual object *Objname2* pointed by the *handles.ObjName2* argument.

In addition, *num2str* is a standard Matlab command, whereas *adcoutcode*(1) *and* *adcoutcode*(2) are typical MEX-C++ input/output functions.

5 Experimentation modes

The starting screen of Matlab GUI/MEX-C++ application presented in Fig. 8, provides a variety of configuration and experimentation modes including Digital-to-digital conversion *(*DDC), digital-to-analog conversion (DAC), analog-to-digital conversion (ADC), and Open/Closed loop tests.

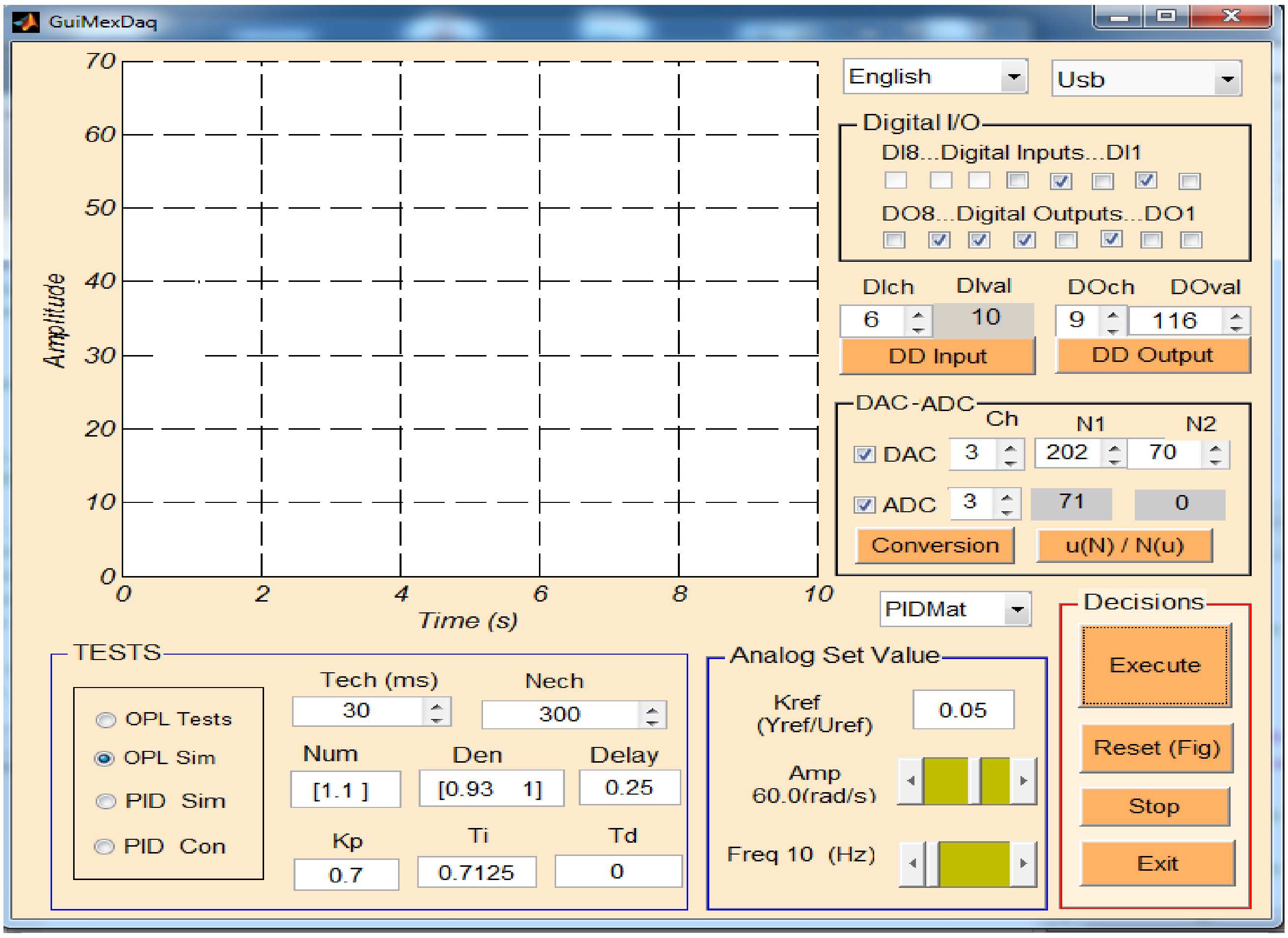


Fig. 8 Starting screen of Matlab GUI/MEX-C++ application software of the workbench

5.1. Digital-to-digital conversion mode

The digital-to-digital conversion (DDC) frame provided in Matlab GUI is presented in Fig.9.

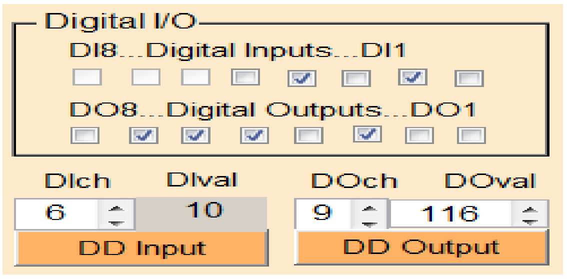


Fig. 9 Digital-to-digital conversion frame

If the *DD Input* control button is clicked, then in the corresponding callback procedure, the input channel state *DIch* is red and displayed as a label object *DIval* according to the function syntax *ddcinp(DIch)*. Otherwise, if the *DD Output* control button is clicked, then the value *DOval* is sent to the digital output according to command syntax *ddcout* (*DOch*, *DOval*).

5.2. DAC/ADC modes

The *DAC/ADC* frame is presented in Fig. 10. In the callback procedure associated with the control button labeled *Conversion*, the MEX function *dacinpcode* (*Ch*, *N1*, *N2*) could be called for DAC, or/and *adcoutcode* (*Ch*) may be called for *ADC* depending on the state of the related checkbox. Recall from Table 1 that *Ch* = 3 (channel number) stands for DAC1 & DAC2, or ADC1 & ADC2.

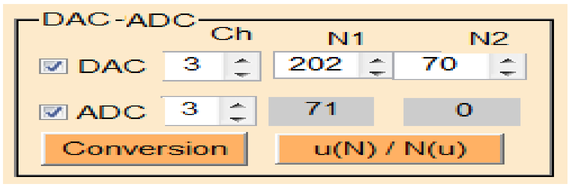


Fig. 10 DAC and ADC frame.

Then, in each case and for each conversion step, a couple of experimental data is collected since a standalone digital voltmeter could be used to measure the external analog value, whereas the related digital value displayed on the *DAC/ADC* frame could be red. At the end of the experiment, the set consisting of coupled of experimental data, can be recorded in the current folder as a Matlab array for direct reading and plotting from a callback procedure associated with the control button labeled *U(N)/N(u)* in Fig. 10. The manual record of DAC/ADC data might be automated in next version of the proposed software.

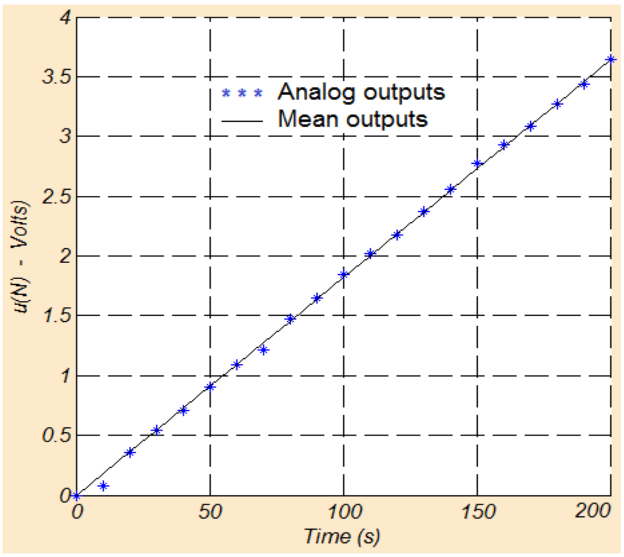


Fig. 11 Input-output characteristic of DAC channel

Fig. 11 shows a typical input-output characteristic of a single DAC channel with mean values modeled as follows:

$$u(N) = 0.0201\ N \qquad (1)$$

where *N* stands for the integer input code.

5.3. Open/Closed loop tests mode

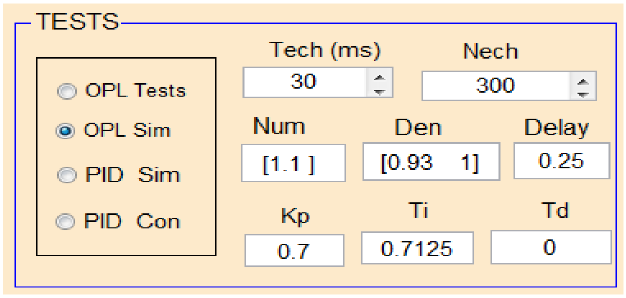
The *Open/Closed loop* tests frame is presented in Fig. 11, where the set of modifiable input data is :

- ✓ *Tech*: Sampling period in milliseconds
- ✓ *Nech*: Number of sample
- ✓ *Num* = *K*: Numerator of the plant transfer function $G_c(p)$.
- ✓ *Delay* = λ: Time delay of the dynamic plant
- ✓ *Den* : Denominator of $G_c(p)$.
- ✓ K_p, T_i, T_d: Parameters of PID (proportional, Inegral and derivative) controller.

As shown in Fig. 12, four main varieties of tests can be conducted by the user from the *Open/Closed loop* tests frame.

5.3.1 Open loop (OPL) Tests

The step response with speed as output is conducted in this case, using a callback of the MEX-C++


TESTS

OPL Tests [✓ OPL Sim] PID Sim PID Con

Tech (ms)	Nech	
30	300	

Num	Den	Delay
[1.1]	[0.93 1]	0.25

Kp	Ti	Td
0.7	0.7125	0

Fig. 12: Open/Closed loop tests frame.

command *oploco.Mexh32* described earlier in Table 1. Then, the experimental data related to speed values are acquired, plotted and saved into the disk for further estimation of a dynamic model of the servo system.

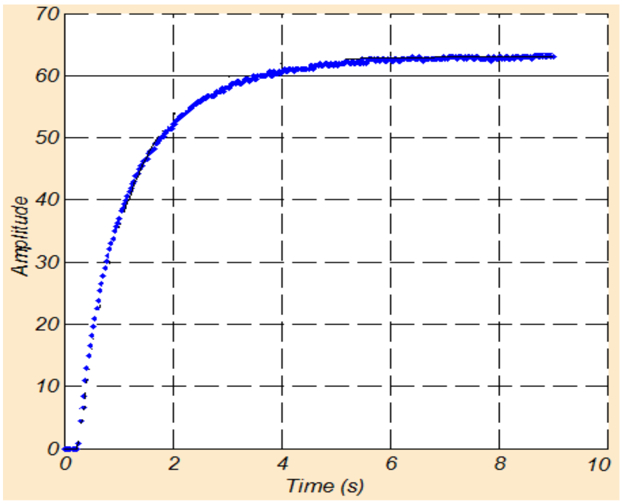


Fig. 13 Test results for open loop step response.

Given the visual properties observed on the experimental step response plotted in Fig. 13, the dynamic behavior of the servo system could be modeled as follows using a first order transfer function with delay:

$$G_c(s) = \frac{Y(s)}{U(s)} = \frac{K}{a_1\ s + a_0} e^{-\lambda\ s} \qquad (2)$$

where K_s, λ , a_1 and a_0 are parameters to be estimated offline using an identification method.

5.3.2 Open loop Simulation

Using Matlab Identification toolbox [13], the parameters K_s, λ , a_1 and a_0 are estimated given experimental data and the model structure (2). As a result, the default values of these parameters are displayed in Fig. 12, in which case the nominal transfer function considered is given as follows:

$$G_c(s) = \frac{1.1}{0.93\ s + 1} e^{-0.25\ s} \qquad (3)$$

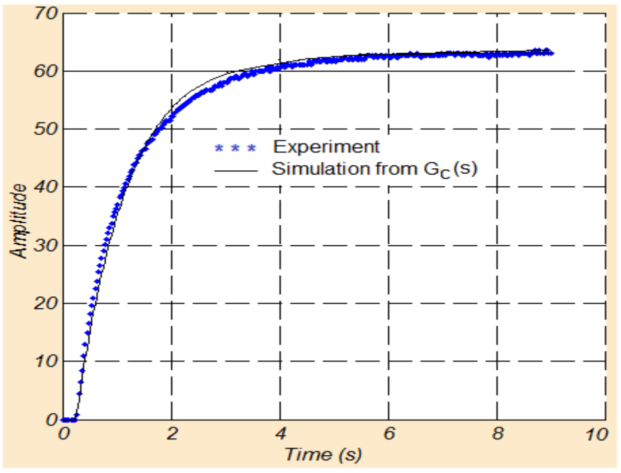


Fig. 14 Open Loop step response (simulation)

In Fig. 14, the simulated open loop step response of is successfully compared to the experimental test.

5.3.3 PID Control Simulation

Given the dynamic model (3) of the servo system, the parameters of the PID controller to be used to

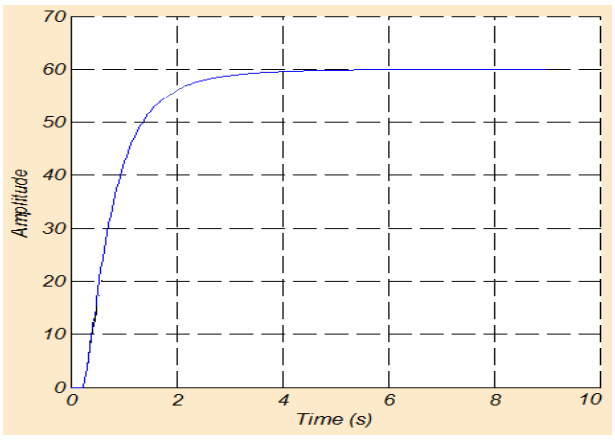


Fig. 15 Simulation results for PID control of speed.

achieve good performance under digital control, could be designed offline using any method available in control engineering ([14]-[18]). The default parameters of the PID transfer function

$$D_c(s) = \frac{U(s)}{\in(s)} = K_p + \frac{1}{T_d\ s} + T_i\ s \qquad (4)$$

are provided on Fig. 11, and the simulation result obtained under a step input is presented in Fig. 14.

5.3.4 Digital PID control and robustness

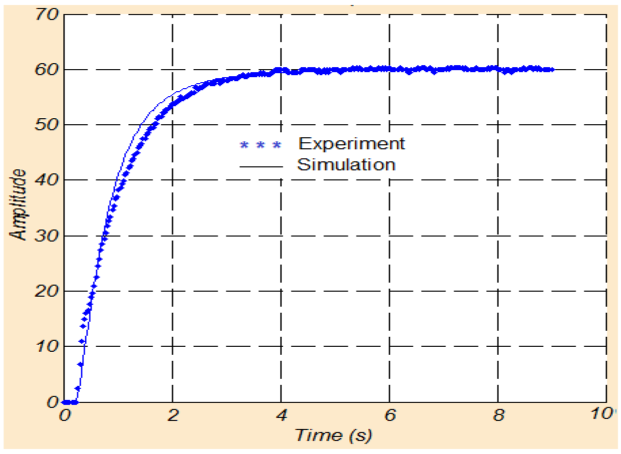
The digital PID algorithm obtained from (4) using Tustin's technique $s \rightarrow \frac{2}{T}\left(\frac{z-1}{z+1}\right) = \frac{2}{T}\left(\frac{1-z^{-1}}{1+z^{-1}}\right)$ [19], is given by (5) given (6).

$$u(k) = u(k-1) + a_0 e(k) + a_1\ e(k-1) + a_2 e(k-2) \quad (5)$$

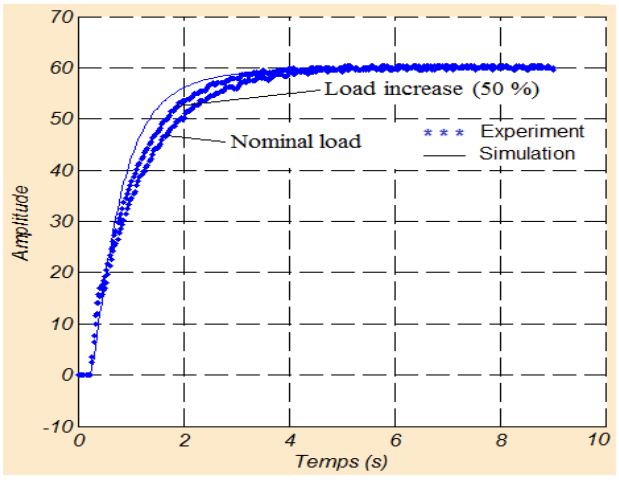
$$\begin{cases} a_0 = K_p\ \left(1 + 2\frac{T_d}{T} + \frac{T}{2\ T_i}\right), \\ a_1 = K_p\ \left(\frac{T}{T_i} - 4\frac{T_d}{T}\right), \\ a_2 = K_p\ \left(\frac{T}{2\ T_i} + 2\frac{T_d}{T} - 1\right) \end{cases} \qquad (6)$$

Following the novel MEX-C++ library summarized in Table 1, recall that (5) given (6) is implemented as optional Matlab and MEX controllers.

The experimental closed loop behavior of the servo system is shown in Fig. 16. In Fig. 16(a) the emphasis is on PID control under the nominal (or default) parameters on the plant, whereas the results shown in Fig. 16(b) are obtained under 50 % variation of a load disturbance, consisting of a mobile magnetic brake.

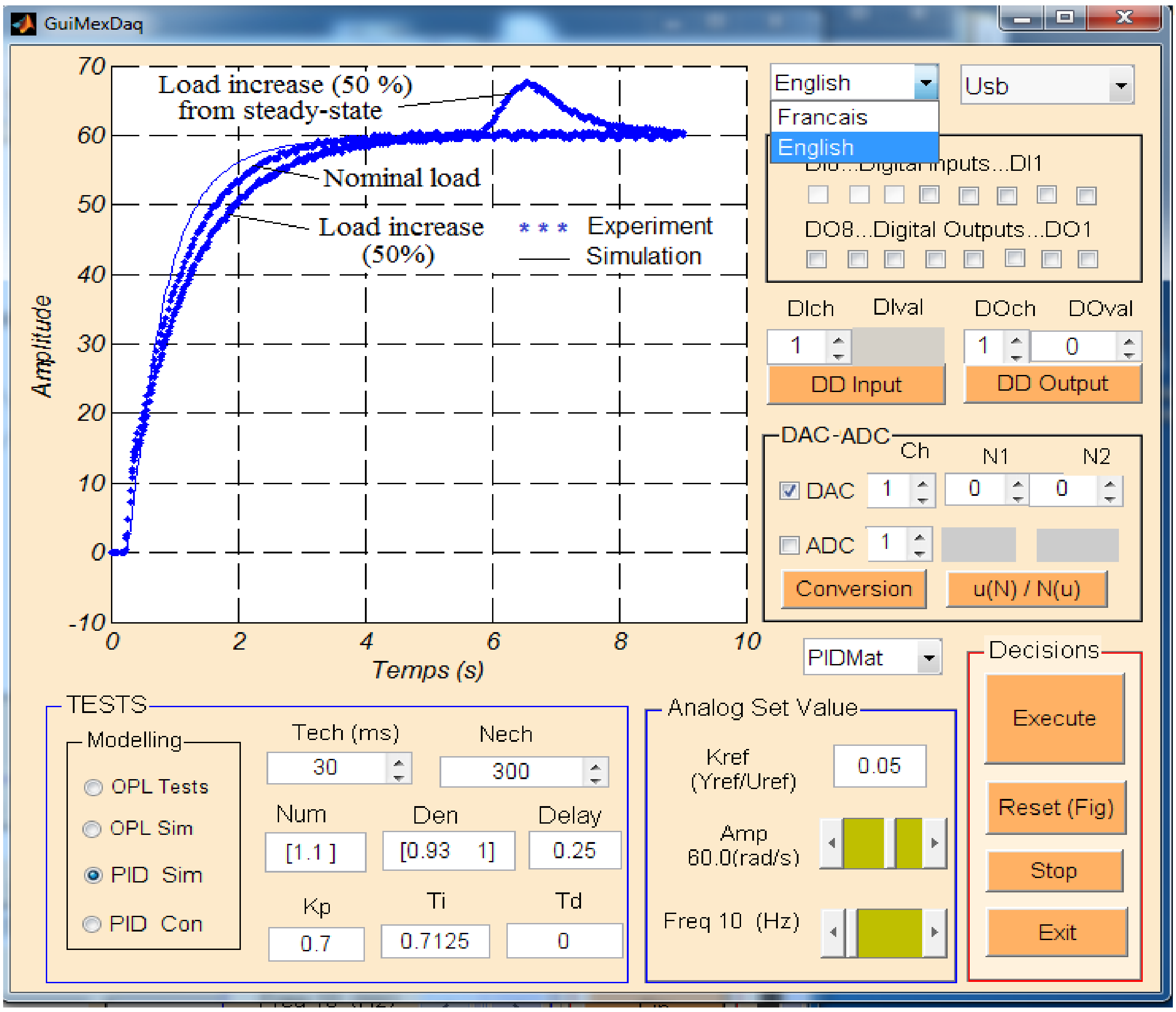


(a) Test results for PID control of speed.



(b) Test results for control robustness of speed.

Fig. 16 Test results - PID control and robustness.



Language: English; Usb; DIch 1; DOch 1; DOval 0; DAC [✓] Ch 1, N1 0, N2 0; ADC [ ] Ch 1; PIDMat

Modelling: OPL Tests, OPL Sim, [✓ PID Sim], PID Con

Tech (ms) 30; Nech 300; Num [1.1]; Den [0.93 1]; Delay 0.25; Kp 0.7; Ti 0.7125; Td 0

Kref (Yref/Uref) 0.05; Amp 60.0(rad/s); Freq 10 (Hz)

Fig. 17 Sequential experiments for speed control.

In addition, it is possible also to make sequential experiments, and to plot all graphical results on the same object figure for the sake of better comparison and well understanding. As an example, Fig. 17 shows the results obtained from the following sequential experiments:

a) PID control simulation of the speed;
b) Matlab-PID control under nominal load;
c) Matlab-PID control under 50 % increase of the load disturbance;
d) Matlab-PID control with 50 % increase of load disturbance from steady state at 5.8 s.

5.3.5 Impact of the proposed virtual workbench

In the electrical and electronic engineering department, of the Advanced Teachers' Training College for Technical Education of the university of Douala, the proposed PC-based virtual workbench has become a common versatile platform, for virtual instrumentation and digital control courses, given its rich palette of ready-to-use simulation and experimentation tools.

For undergraduate students, laboratory work sessions needs beside Matlab software, the preinstalled custom resources (in each Laptop to be used as a virtual instrument), consisting of:

a) Matlab main program *MexGuiDaq.m*;
b) Matlab GUI application *MexGuiDad.Fig*;
c) Compiled *MEX-C++* library (*.MexW32);
d) C++ driver K8055.Dll of the DAQ target.

Matlab GUI application *MexGuiDad.Fig* is activated automatically when running the main program *MexGuiDaq.m* from Matlab prompt. In this mode, visual objects and their appearance on Matlab GUI, cannot be neither updated nor modified by the user. Following our own experience, it is important to mention that, under minimum help of an instructor, a mean of 30 minutes is sufficient for undergraduate students to understand and setup equipments of the workbench, and to start Maltab GUI/MEX application. In addition, each experiment selected from Matlab GUI application is usually conducted before 5 minutes. Furthermore, a number of independent experiments can be performed sequentially using simple mouse clicks after minor/major changes on target modifiable parameters (if any). In which case updating the physical architecture of the whole workbench, is made automatically and instantaneously. For each experimentation strategy, a screen view of the results obtained can be captured and pasted to a word or image editor, for further use in a technical report to be edited as a homework and submitted to the instructor a few days later for evaluation.

For graduate students, the main aim behind laboratory work sessions using the proposed PC-based virtual workbench, is to enable them to understand in depth the secrets of creating custom virtual instruments and automated systems from Matlab GUI/MEX-C++ programming fundamentals. In this case, beside Matlab, the complete preinstalled custom resources (in each Laptop to be used as a prototyping PC-based virtual instrument) consist of:

a) Matlab main program *MexGuiDaq.m*;
b) Matlab GUI application *MexGuiDad.Fig*;
c) Compiled *MEX-C++* library (*.MexW32).
d) MEX-C++ library source codes (*.CPP);
e) C++ driver K8055.Dll of the DAQ target.
f) Standard C++ development tool e.g. Bloodshed Dev-C++ if any, otherwise the default Matlab C++ compiler might be used along with a text editor for viewing or modifying MEX C++ codes.

Subsequently, both Matlab GUI application *MexGuiDad.Fig* and Matlab main program *MexGuiDaq.m* are activated automatically when running the *guide* command from Matlab prompt, followed by the specification of the target folder and executable file in a *guide* form. In this design mode, visual objects and their appearance on Matlab GUI can be either updated or modified according to the user needs. For each single experiment, graduate students with good prerequisites learned from the theoretical advanced programming course, can open related GUI and MEX source codes, in order to discover and understand with minimum help from the instructor, how real time programming tasks involved are organized, implemented and compiled.

Thus, compared to traditional laboratory sessions where a variety of real instruments and equipments should be frequently reconnected, manipulated and updated by learners, the proposed virtual workbench offers a number of potential merits for both undergraduate and graduate cases, including, significant time saving, higher reliability, better working comfort, greater considerable didactic efficiency, and more powerful base of realistic case studies for engineering researches.

6 Conclusion

The research work presented in this paper, shows how an advanced virtual instrumentation and automatic control tool for PC-based servo systems, can be developed using a mix of Matlab GUI programming strategies and MEX-C++ developing technologies. While Matlab GUI programming has proved to be very attractive for rapid development of visual applications under Matlab platform, the major merit of MEX-C++ development relies on the great opportunity of building sophisticated real time Matlab libraries from C++ drivers of an arbitrary DAQ.

Compared to equivalent codes implemented using standard Matlab commands, MEX-C++ functions runs very fast. In addition, a MEX-C++ library for real time instrumentation and control, might be used under Matlab to drive an arbitrary DAQ board from its basic C++ driver, even though a corresponding driver version for Matlab is unavailable in Matlab DAQ toolbox.

Although the proposed virtual instrumentation and control platform for servo system has proved to be very satisfactory for engineering education, next editions should bring significant improvements and extensions. For example, the acquisition of *DAC/ADC* characteristics might become an online callback decision. In addition, the dynamic model of the servo plan might be estimated from the open loop test, using an automated callback estimation process. Furthermore, it would be fruitful to provide an additional position control option of the servo shaft on the proposed Matlab GUI/MEX-C++ application. These important improvements will be investigated in future research works.

Acknowledgment

The author of this paper sincerely thanks the anonymous reviewers of WSEAS transaction on advances in engineering education, for their very helpful comments and suggestions.

References

[1] A. Munjiza, N.W.M. John, “Virtual Nanotechnology Workbench for Engineering Education”, *International Conference on Engineering Education* – ICEE, September 3-7, Coimbra, Portugal, 2007.

[2] L. Benetazzo, M. Bertocco, F. Ferraris, and A. Ferrero, “A Web-based distributed virtual educational laboratory”, *Proceedings of the 16th IEEE Conference on Instrumentation and Measurement Technology*, Volume:3, pp. 1851-1856, 24 May 1999-26 May 1999.

[3] J. Mbihi and Alexis Motto, “Informatique et automation – Automatisme programmable contrôlés par ordinateur”, 358 pages, 2006, © *Ellipses Editions*, Paris.

[4] M.K. Abuzalata, M.A.K. Alia, Shebel Asad and Mazouz Salahat, “Design of a Virtual PLC Using Lab View”, *Research Journal of Applied Sciences, Engineering and Technology*, Vol 2, No 3, pp. 283-288, May 10, 2010.

[5] J. Mbihi, “Contribution à l’étude et au prototypage d’un banc d’essais didactique flexible d’instrumentation virtuelle et d’asservissement par ordinateur” , Journal sur l’enseignement des sciences et des technologies de l’information et des systèmes, J3EA-Vol. 9, No. 1, 2010, © EDP Sciences.

[6] R. Sethunadh, S. Athuladevi and S. Sankara Iyer, “Virtual Instrumentation Techniques in Test and Evaluation of Launch Vehicle Avionics”, *Defence Science Journal.* Vol. *52.* No. 4, October 2002. pp. 357-362, O 2002, DESIDOC.

[7] S. Antonios Andreatos and D. Anastasios zagorianos, “Matlab GUI application for teaching control systems”, *proceedings of the 6th WSEAS international conference on engineering education*, 208-211, ISBN: 978-960-474-100-7

[8] H. Ali Assi, H. Maitha A. Shamizi and H. N. Hassan Hejasse, “Matlab GUI application for teaching electronics - Engineering education and research using Matlab”, October 2011, © InTech, www.interchopen.com.

[9] J. Mbihi and A. Motto, “Instrumentation virtuelle assistée par ordinateur - Principes et techniques, cours et exercices corrigés”, *Ellipses Editions*, 240 pages, October 2012, Paris.

[10] Zhengmao Ye, Habib Mohamadian, Hang Yin, Guoping Zhang, Su-Seng Pang, “Advancing laboratory education in control engineering with practical implementation approaches”, WSEAS *Transactions on advances in engineering education*, pp. 55-65, Issue 2, Volume 6, February 2009.

[11] Mathworks, “Matlab 2013 – DAQ Toolbox”, www.mathworks.com/products/daq.

[12] Mathworks, “MATLAB Support Package for Velleman K8055/VM110 User Guide ”, June 13, 2011.

[13] Mathworks, “Matlab R2013a - System identification toolbox”, Mathworks.

[14] H. O. Bansal, R. Sharma, P. R. Shreeraman, “PID Controller Tuning Techniques: A Review”, *Journal of Control Engineering and Technology*, Vol. 2, Issue 4, October, pp. 168-176, © World Academic Publishing 2012, www.ijcet.org.

[15] S. N. Deepa and G. Sugumaran, “Design of PID controller for higher order continuous system using MPSO based model formulation technique”, *International journal of electrical and electronics engineering,* Issue 4, vol. 5, pp. 289-295, 2011.

[16] M. Y. Tabari, A. V. Kamyad, “Design optimal fractional PID controller for DC motor with generic algorithm”, *International Journal of Scientific and engineering research*, Vol. 3, Issue 12, pp. 01-04, December 2012, www.ijser.org.

[17] Ziegler, J. G. and N. B. Nichols, “Optimum Settings for Au tomatic Controllers”, *Transactions ASME*, Vol. 64, pp. 758-768 (1942), www.ijsrp.org.

[18] S. Das, A. Chakraborty, J. K. Ray, S Bhattacharjee, and B. Neogi, “Study on Different Tuning Approach with Incorporation of Simulation Aspect for Z-N (Ziegler - Nichols) Rules”, I*nternational Journal of Scientific and Research Publications*, Volume 2, Issue 8, August 2012.

[19] G. F. Franklein, J. D. Powell and M. L. Workman, *Digital control of dynamic systems, 2nd Edition,* Addison-Wesley, 1990.