

Application of xPC Target as a Prototyping Environment in Control System Laboratories

**Cedric Demers-Roy, Richard Hurteau
Ecole Polytechnique de Montreal**

Abstract

This paper presents an example of application of xPC Target (product integrated with Matlab/Simulink) in an educational laboratory for real-time control experimentation. Equivalence between this environment and LabVIEW, another well known real-time environment for control, is discussed. Operating characteristics, cost, realization time, required expertise and implementation time are also presented. This example is realized on a classical system for position control using a DC motor. A complete control design process is applied on this test bench, from simulation to real-time control implementation. Result analysis indicates that xPC Target is a powerful environment for fast prototyping of real-time control loop as needed by undergraduate laboratories and for research activities implying fast testing of control algorithms on a specific system.

I. Introduction

Experimental testing of control algorithms requires an environment where simulation analysis and real time testing can be appropriately and efficiently realized. Unfortunately, simulations are time consuming and, most of the time in class room context, the effort needed to transfer the code from the simulation environment to the real-time control environment is underestimated, leading to poor results analysis. In both industrial and academic fields, development time is critical and it is important to maintain the development effort minimal. Ideally, simulation and real-time implementation of control algorithms on a given system should be realized on the same development environment, so as to minimize this development time. To attain this objective, the selection of an environment for both simulations and real-time control is a critical choice.

Commercial software solutions like xPC Target (with Matlab/Simulink) and LabVIEW offer integrated environment for simulation and real time implementation. Both include tools to help reduce time and effort spent on the development of a module for simulations and real-time implementation of a given control theory. Moreover, a survey done in 1999 by ITRON [1] shows that performance, proved track record, industry usage and reliability are the main criteria for the selection of an operating system for a given control engineering project. In this paper Matlab and LabVIEW are explored as an integrated environment since they are commonly used

in industrial and academic context and they show good performance at first glance. Matlab is mainly used in research and education context for simulations on mathematical models, and LabVIEW is mainly used in real-time control in industrial context. Both products offer the tools to integrate simulations and real-time implementation.

II. Test bench for position control

To explore both environments, a physical system will be considered: a test bench for position control using a DC motor and a single loop control commonly found in various applications. The first version of this test bench was presented in [6]. The objective is having the same environment for design, implementation and data analysis. Since the ultimate goal is to reduce the time spent on the design phase, the environment needs to be easy to use in both design and implementation phases.

The actual test bench in our laboratory uses LabVIEW as the control environment and xPC Target is proposed as a new alternative environment. Figure 1 illustrates the physical setup of this test bench:

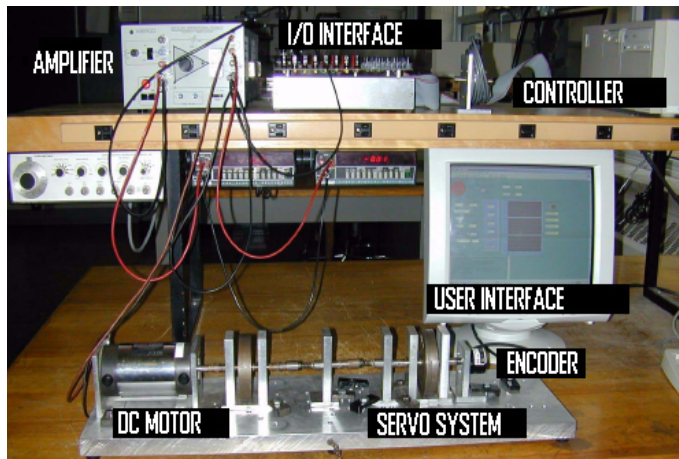


Figure 1: Test bench components and physical setup

- a **servo system** consisting of a **DC motor**, two inertia loads, a mechanical link and an output encoder;
- a **controller** and **interface** running on a desktop computer (two computers in the case of xPC Target);
- an **I/O** (inputs / outputs) **interface** box for connections between the computer and the system;
- an **amplifier** for power conversion of the command coming from the computer;
- two voltmeters (not highlighted on figure) to help monitoring system state during control loop execution.

Modifications can be made on the servo system. The mechanical link between the two inertia loads can be a rigid link (direct coupling) or elastic link (torsion springs). Figure 1 show the torsion spring in the center of the bench. Complete equipment for this system is:

- one fast (1 GHz)PC operated under Windows (2000, XP or similar) and with:
 - Matlab 7.0.1 (with xPC Target 2.6.1);
 - SVGA monitor for design and operation;
- one older PC (133 MHz) or any Intel compatible PC with any or no OS and with:
 - Ethernet TCP/IP card (xPC Target compatible, like Intel[®]PRO-100S) ;
 - National Instrument PCI-6025E analog I/O card (to output command);
 - Measurement Computing CIO-QUAD02 encoder input card (2 channels);
 - SVGA monitor is optional but can be useful sometimes;

- one KEPCO BOP-36M operational amplifier (± 36 VDC; 5 A; max gain of -9.9);
- one I/O interface box for connections between target PC and the system;
- two digital voltmeters are optional but useful for system tuning;
- physical system:
 - DC motor ;
 - end-line optical shaft encoder with 200 pulses per turn and two output channels for quadrature modes;
 - inertia loads (wheels) with a radius of 50 mm and a mass of 1 kg;
 - elastic link (homemade torsion springs).

The open loop model for this system (considering the second inertia wheel position as the output) is given by the transfer function of equation 1. An open loop test and a closed loop test using the Haglund Astrom non linear method (high gain proportional controller with saturation of the command at 1 Volt) are necessary to identify the parameter values. The optimal parameter values are then obtained using function minimization (*fminsearch* in Matlab) and the mathematical equations linking the model and the actual tests results. The complete procedure and equations can be found in [7].

$$\frac{\Phi(s)}{U(s)} = \frac{G}{s(s^2 + as + b)(s + c)} \quad (1)$$

with $a=3.5$ sec, $b=177$, $c=4$ sec⁻¹ and $G=8126$ rad/sec*volt.

To control this system a novel PID configuration is used. This novel PID is named “PID-Dual-Loop” (later referenced as PIDDL in this paper) and has been proposed by DeSantis [8]. One of the main advantage of this controller is that the tuning of its gains is realized independently for the follower mode (reference following) and regulator mode (disturbance cancelling). The actual mathematical form of this controller is given by equation 2. Complete procedure for experiment with this controller can be found in [].

$$U(s) = K_1 [R(s) - Y(s)] - K_2 sY(s) + K_3 \left[\frac{R(s) - Y(s)}{s} - \alpha_1 Y(s) - \alpha_2 sY(s) \right] + K_4 R(s) \quad (2)$$

with R the reference, Y the output, U the command;

$K_1, K_2, K_3, K_4, \alpha_1$ and α_2 the controller parameters.

To help obtain the best performances on the system using the elastic link, a « masking » filter is added to cancel the main elastic poles of the transfer function (equation 1). Equation 3 gives the transfer function of this filter, which is inserted between the controller and the system.

$$\frac{U_o(s)}{U_i(s)} = \frac{p^2(s^2 + as + b)}{b(s + p)^2} \quad (3)$$

with a and b as identified in simulation model; p as the new double pole.

Since testing any algorithm always implies running both simulations and real-time control tests, the complete application program includes the simulation model and the real-time control model. A logical switch will allow the activation of each model in the final implementation. The same logical activation procedure is done for each component of the model: the controller can be de-activated (resulting in an open loop operation), the filter can be de-activated ($U_o/U_i = 1$ in this case) and the rigid link model can be selected for simulation instead of the elastic link model.

III. LabVIEW implementation of the real-time controller

The version 7.0 of LabVIEW (see technical reference on *National Instruments* website [5]) used in the case study operates under Windows XP SP2. To obtain the best performance with Windows, which is not a real time operating system, the VI (Virtual Instrument) programmed for the case study, is set on *time critical priority* in the *execution* part of its *VI properties*.

A Virtual Instrument in LabVIEW consists of a graphical panel grouping controls and a circuit-like diagram grouping functional blocks. While designing an application is easy and fast with this system, designing a complete interface for parameter tuning and data logging on a complete system needs more time and special attentions. For the position control, the tabbing option of the graphical interface was used (see figure 4) and the resulting diagram is quite complex. Overall diagram for this control application spans on many computer screens in its actual state (see figure 5 for a small portion of the “while” control loop).

Tools available in LabVIEW are primarily designed to be used in-line with the execution of the program. Data logging was chosen to allow post-execution analysis. Analysis of the logged data is realized using Matlab. Since Matlab is already well-known for modelling and simulation, it was natural to use it for the data analysis. Data logging is realized by filling a table with execution data, and then saving it in a text file using the corresponding I/O blocks available in LabVIEW. In this text file, few command lines are added to make it directly compatible with Matlab. The block *create tab pid* at bottom right of figure 5 manage this table.

To create the real-time system I/O, a block named *DAQ Assistant* simplify the integration and configuration of the I/O card present in the computer. Data formatting is then done using common functional blocks.

Figures 4 and 5 shows the actual implementation of this system in LabVIEW with the interfaces needed to modify the reference, the sampling period, the PID controller gains, the system type and the system's parameters. Circuit diagram also include a section (not shown on figure) to transfer all useful signals in the text file later analyzed with Matlab.

IV. Matlab-Simulink-xPC target implementation

The 7.0.1 version of Matlab (see technical reference on *The Mathworks* website [3]) used in our example operates under Windows XP SP2. However, Matlab integrates a tool named *Real*

Time Workshop that allows targeting real-time code to *xPC Target*, which is essentially an operating system for the execution of control loop running on a second computer. Matlab also have a graphical extension named Simulink (see technical reference on *The Mathworks* website [2]) which allows modelling of dynamical systems using functional blocks and data flow signals. Since Matlab and Simulink share the same memory workspace, variables in Matlab and Simulink are common to each other. This characteristic eliminates the need for a complete graphical interface (like the LabVIEW panel) to manipulate the various parameters of the system.

For each specific application, a Matlab script and a Simulink model are developed to fulfill the corresponding needs in terms of simulation and design of control law. The interface proposed on figures 6 and 7 has the same role than the one in LabVIEW (figures 4 and 5). Since a script is used for the configuration, the resulting model is very simple and easy to understand and modify. It is also possible to use the *guide* function in Matlab to create a full graphical interface. But the result wouldn't be as simple as the script and the model actually presented. The simulation model is created using simple functional blocks like constants, gains, transfer functions, scopes, summations, etc. Sub-systems are created to group these blocks in functional groups like the controller, the physical system, the reference generator and the visualization interface. The parameters of these blocks are directly read from the Matlab workspace which is initialized by the script. For the I/O of the real physical system, *xPC Target* library contains over 250 driver blocks for various I/O cards available on the market. To use a specific card, the corresponding block in the library must be found and inserted in the model to replace the simulation model (see case study).

Compiling is an essential step using *xPC Target*. Real Time Workshop is called to create a version of the actual Simulink model to be executed on the target computer. The actual configuration of a system using *xPC Target* is of the type host – target, involving the use of two computers. The first computer running under Windows is used only for design and for control execution and the second computer running on the *xPC Target* kernel allows a perfect real-time execution of the compiled model. In the most recent version (*xPC Target* 2.6.1 with Matlab 7.0.1), the compiled model is actually linked to the Simulink model on the host to allow easy control of the execution on the target. This link is activated by the external mode on the Simulink model.

Communications between the host and target use the TCP/IP protocol on an open (WAN) or closed (LAN) network. A closed network (intranet) permits to obtain maximal link speed in our laboratory. The command *xpcsetup* in Matlab allows the user to configure all the parameters of *xPC Target* on both the host and the target. Notice that the target is set up using a boot disk that actually configures the system and launches the kernel from the floppy disk. This allows to take any existing computer and to use it as a target as long as it can be reached by the network connection (some Ethernet card are not supported; in our application the Intel[®]PRO/100S Ethernet card and the I82559 driver on *xPC Target* were used). It is interesting to note that even an old processor like a Pentium 133 MHz can gives good real-time performance (a sampling period of about 1 millisecond) with *xPC Target*.

To analyze the logged data, a specific script is created to store the data in the Matlab workspace allowing an easy analysis and graphical presentation. Since all these operations are realized in the Matlab workspace there is no need to transit the data through a file. When data analysis is done, Matlab offers the possibility to save its workspace in a « .MAT » file that can be stored on a permanent memory and loaded later if necessary.

Globally, xPC Target allows to develop both the simulation and real-time control environment on the same Simulink model, with the configuration and the data analysis realized using Matlab scripts. All these tools use the same workspace which greatly simplifies exchange of information between them, using variables, vectors and matrices. Since xPC Target uses its own operating kernel, a second computer is necessary. However, the boot disk generated by xPC Target converts temporarily any computer as a target for xPC as long as a TCP/IP connection can be established between the host and target PCs.

IV.1 Simulation and real-time implementation using xPC Target

In Matlab/Simulink/xPC Target, the possibility to use configuration scripts and the shared workspace between Simulink and Matlab, allows us to create a very simple diagram with few data flow signals and a sub-system representation, that groups the various blocks of a system in a single block easy to identify and to understand. Figures 6 and 7 illustrate the script used for configuration and the root model of the system. Blocks on figure 7 are in fact sub-systems containing many other blocks. There is no graphical interface on this model since it is easier to modify the configuration script than to build and use a complete graphical panel as in LabVIEW.

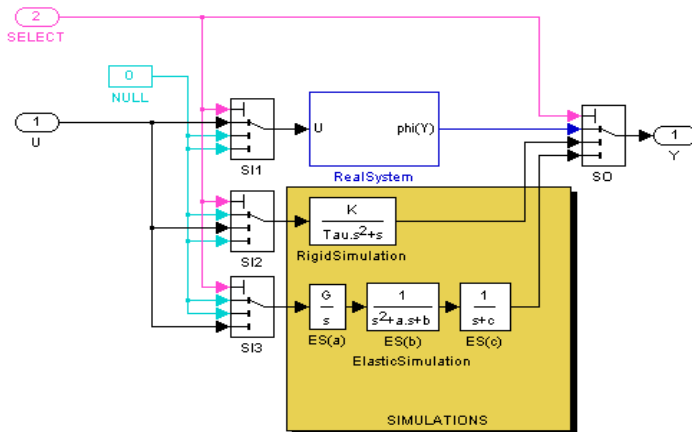


Figure 2: Content of the system block in the Simulink model for first case study

Figure 2 shows the content of the *system* block of figure 7. The highlighted part of this figure shows the models for simulation (rigid and elastic link) and the top part of the figure shows the I/O blocks for the real-time control model. The configuration script groups all the parameters needed to guarantee correct data flow in each mode (rigid or elastic simulation, and real control). Every other sub-systems in the complete diagram of figure 7 uses functional blocks from Simulink library: *constants*, *sum*, *mux*, *demux*,

gain, *transfer function*, *integral*, *derivative*, etc. After the control model is created and all its parameters properly initialized, the build function of Real Time Workshop is called and the control model is compiled (xPC Target need to be configured as explained in the technical documentation from *The Mathworks* [4]).

Execution on the target should always pass by the host version of the application to allow parameter modifications between each execution of the control loop. To synchronize both models (host and target versions), xPC Target uses the *External* mode of Simulink. When they are connected together, the parameters on the host and on the target are synchronized. This *External* mode allows fast testing for sets of parameters without compilation of the model.

Once the execution on the target system is completed, simple commands allow downloading the resulting data on the host system by the Ethernet link. Two different logs are useful: the *outputlog* and the *timelog*. The output-log contains every signal the user wants to memorize. A third log command can be used to track system's performances: the task execution time log (TET log). This log memorizes the time used at every sample to calculate the control law of the model on the target. Larger is the difference between the TET and the sampling period, easier it will be to control the execution, since more time will be available for communication functions of the xPC Target operating system.

When all data have been downloaded from the target to the host, data analysis is realized with the standard functions of Matlab / Simulink, including the *save* or *uisave* functions for storage of the data. Since the workspace between Matlab and Simulink is completely shared, no added data links are needed on the model and all the data processing is automated using a script.

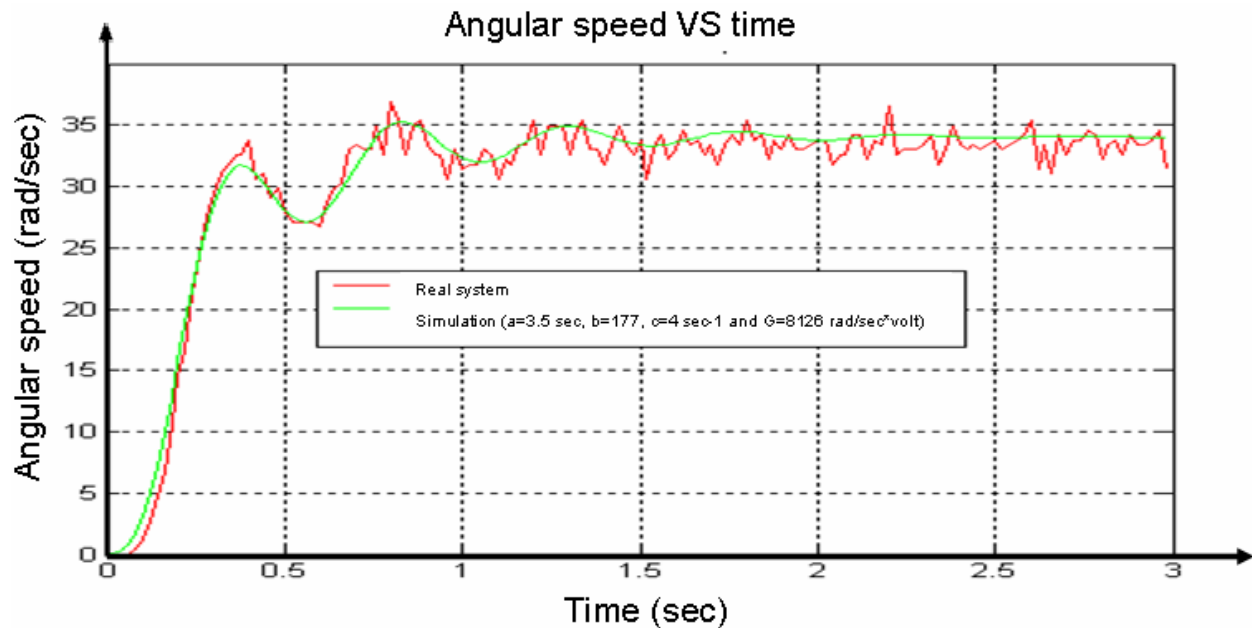


Figure 3: Actual results for a test in simulation and real control (using proposed system model)

Figure 3 shows results where simulation mode and real-time mode results are superposed. These results are obtained by two executions of the model presented on figure 6 (with only a change on the *ex_mode* parameter). This parameter controls the switching between the real system I/O, the rigid link simulation model and the elastic link simulation model (see figure 2). With the external mode in Simulink, it is even possible to change configuration parameters

without compiling the model again. This allows fast testing of the controlling algorithm actually implemented. To test a new algorithm, only the controller block in the model and the configuration script have to be modified (no compiling is needed again).

All these features give to our undergraduate laboratories a new way to develop more interesting and motivating demonstrations and allow the introduction of new algorithms that were not programmed experimentally before. Since the final diagram is very simple in Simulink and the scripts in Matlab are easy to modify, changes are easy and fast to program.

Another feature that has been successfully tested in our laboratories is the capability to drive an xPC Target application from a second model running in Simulink. This scheme allows the implementation of a two level controller. For example, xPC Target can run a model similar to the one in the case study (see figure 7), but receives its parameters from a joystick input (instead of the configuration script) acquired through a second Simulink model (Joystick input block from the *Virtual Reality* library). In this case, buttons on the joystick allow to switch signal values and the joystick axis input could become the reference signal of the controller. This allows experiments with dynamically changing signals and parameters.

V. Simulation and Real-time performances

On this classical control example, both applications developed with LabVIEW and Matlab / Simulink / xPC Target gives the same simulation results and similar real-time results for a relatively large sampling period (20 ms). In terms of development time, figures 4 and 5 (LabVIEW) compared to figures 6 and 7 (xPC Target) shows the difference in complexity resulting from both programming structures. The shared workspace in Matlab allows fast and easy parameter changes and Simulink diagrams are also easy to modify. In LabVIEW, modifications are more time consuming and every parameter change must be carefully planned and included properly in the panel. However both environments offer essentially the same tools and the resulting model is similar. In this case, parameters such as ease of use, available capabilities, implementation time, and background expertise required are too qualitative to be used for a discussion around performances of these two environments. As for the acquisition cost, both products are similar with a few thousands dollars for the most common options in an industrial context, and both products offer a discount for academic use.

For our needs in real-time control, operating characteristics like sampling time and time stability are quantitative data important to analyze. The first parameter is about the smallest sampling time the system can operate, and the second is about how constant and robust is the application of this sample time. For the numerical computations needed in the controller, one CPU is used for execution in LabVIEW and xPC Target, but development and execution control can be done on a second computer in the xPC Target case, while these functions are supported by the same CPU in LabVIEW.

With LabVIEW, a sampling period of 1 ms on a 266 MHz processor gives results shown on figure 9. This figure shows that it is not possible to obtain the sampling period specified (mean sampling time of 15 ms). To obtain best performances with LabVIEW, an extension

called LabVIEW Real Time is needed, and it includes a specific processor (more expensive option) for targeting the real-time application. Because xPC Target uses any personal computer as a target, there is an advantage in terms of simplicity and performances for rapid prototyping.

To illustrate performances of xPC Target, a time log file is analyzed. Since it contains sampling times, difference between two samples of this log is the elapsed time between two samples. This information and the content of the TET (Task Execution Time) log show xPC Target kernel performances. This test is done with a sampling time of 1 millisecond (ms) in three different execution conditions:

- 1) simulation mode with elastic link model;
- 2) simulation mode with rigid link model;
- 3) real-time control mode (elastic link on physical system).

On figure 8 there is absolutely no glitch on the curves of sampling time versus time. This shows good stability of the sampling time. The same test can be done using a 500 microseconds (μs) or even 250 μs sampling period. A function in xPC Target, *xpcbench* ('this'), automatically puts the target computer under test to determine its performance. The results of this test shows a minimal sampling time ranging from 12 μs (for a very simple model) to 819 μs (for a very complex model) using the 266 MHz processor of the computer used for the case study. Automatic detection of sampling period overrun is also included in XPC Target to ensure robust real-time execution.

VI. Conclusion

In this paper, xPC Target has been presented as a development and operating environment for real-time control experimentation. A classical test bench for position control has been used to implement a real-time controller with LabView and xPC Target. Both implementations give satisfactory results. However the Matlab/Simulink/xPC Target combination can clearly be used as an environment for fast prototyping and testing of control algorithms in both simulation and real-time control modes. For undergraduate control laboratory applications, these characteristics are useful to obtain more motivating demonstrations and more interesting and challenging projects. Modelling, simulations and real-time control in this environment is a fast process, easy to develop or to modify, and Matlab scripts allow quick configuration and easy data analysis.

Because of its performances and easy to use, xPC Target has been adopted in other projects in our laboratories: design and implementation of a vehicle for rapid prototyping of intelligent guidance algorithms [9], assisted guidance for a tractor-trailer like vehicle [10] and [11] and temperature control for a three heat zone cylindrical oven [12]. Other projects are still to come with new versions of the software.

Acknowledgments

The authors want to thank professor Romano DeSantis for his contribution to this work and his helpful comments on this paper.

Bibliography

- [1] ITRON, *Survey Results on Real-Time OS Use Trends and ITRON*, [En ligne].
<http://www.na.assoc.tron.org/index.html>
(Results of survey not available anymore on 11/29/2004)
- [2] Mathworks, *The MathWorks – Products – Simulink – Simulation and model-based design*, [En ligne].
<http://www.mathworks.com/products/simulink/index.html>
(Not available anymore on 5/1/2005).
- [3] Mathworks, *The MathWorks – Real Time Workshop – Generate optimized, portable, and customizable code from Simulink models*, [En ligne].
<http://www.mathworks.com/products/rtw/description1.shtml>
(Verified on 5/1/2005).
- [4] MathWorks, *The MathWorks – xPC Target – Perform real-time rapid prototyping and hardware-in-the-loop simulation using PC hardware*, [En ligne].
<http://www.mathworks.com/products/xpctarget/index.html>
(Verified on 5/1/2005).
- [5] National Instruments, *Acquire, Analyze, and Present Data with LabVIEW – Products and Services – National Instruments*, [En ligne].
<http://www.ni.com/aap/>
(Verified on 5/1/2005).
- [6] DeSantis, R.M, Hurteau, R., Authié, G., A Real-Time Computer Control Demonstrator for Use in a University Control Laboratory, **IEEE Transactions on education**, vol e-25, no1, 1982, pp 18-28.
- [7] Hurteau, R., DeSantis, R.M., **Cours ELE3201 – Guide des laboratoires – Hiver 2004**, Rapport interne, Département de génie électrique, École Polytechnique de Montréal, 2004, 66 pp.
- [8] DeSantis, R.M, A Novel PID Configuration for Speed and Position Control, **Journal of Dynamic Systems, Measurement, and Control**, Transactions of the ASME, vol 116, 1994, pp 542-549.
- [9] Beaudry, J., Demers-Roy, C., Un véhicule robotisé pour le prototypage rapide de stratégies de guidage intelligent, Département de génie électrique, École Polytechnique de Montréal EPM/RT-2003-14, Montréal, 2003, 27 pp.
- [10] Demers-Roy, C., Mirales, F., Olivier, J.-F., De Santis, R.M., Hurteau, R., Assisted Guidance For a Tractor-Trailer With Off Axle Hitching, **5th IFAC Symposium on Intelligent Autonomous Vehicles**, Lisbon, july 2004, 5 pp.

- [11] Demers-Roy, C., Miralles, F., Olivier, J.-F., Guidage assisté par ordinateur d'un tracteur remorque, Département de génie électrique, École Polytechnique de Montréal EPM/RT-2003-08, Montréal, 2003, 27 pp.
- [12] Blanc, L., Rainbault, V., Contrôle en température d'un four cylindrique à trois zones de chauffe, Département de génie électrique, École Polytechnique de Montréal EPM/RT-2003-08, Montréal, 2003, 38 pp.

Biography



Cedric Demers-Roy received a B. Sc. A. degree in computer engineering and the M.S. degree in Electrical Engineering from the École Polytechnique de Montréal (Canada) in 2001 and 2004 respectively. For the last three years he has been in the Department of Electrical Engineering at the École Polytechnique de Montréal as a professor assistant in both classroom and laboratory teaching. His interests concern principally applied control and real time system applications of control theory.



Richard Hurteau received a B. Sc. A. degree in electrical engineering from École Polytechnique de Montréal (Canada) in 1974 and the Docteur-Ingénieur degree in Control Systems from Université Paul Sabatier, Toulouse, France in 1977. Since 1977, he is a professor with the Department of Electrical Engineering at École Polytechnique de Montréal and since 2003 he is Head of the Department of Electrical Engineering. His interests concern real-time control and robotics.

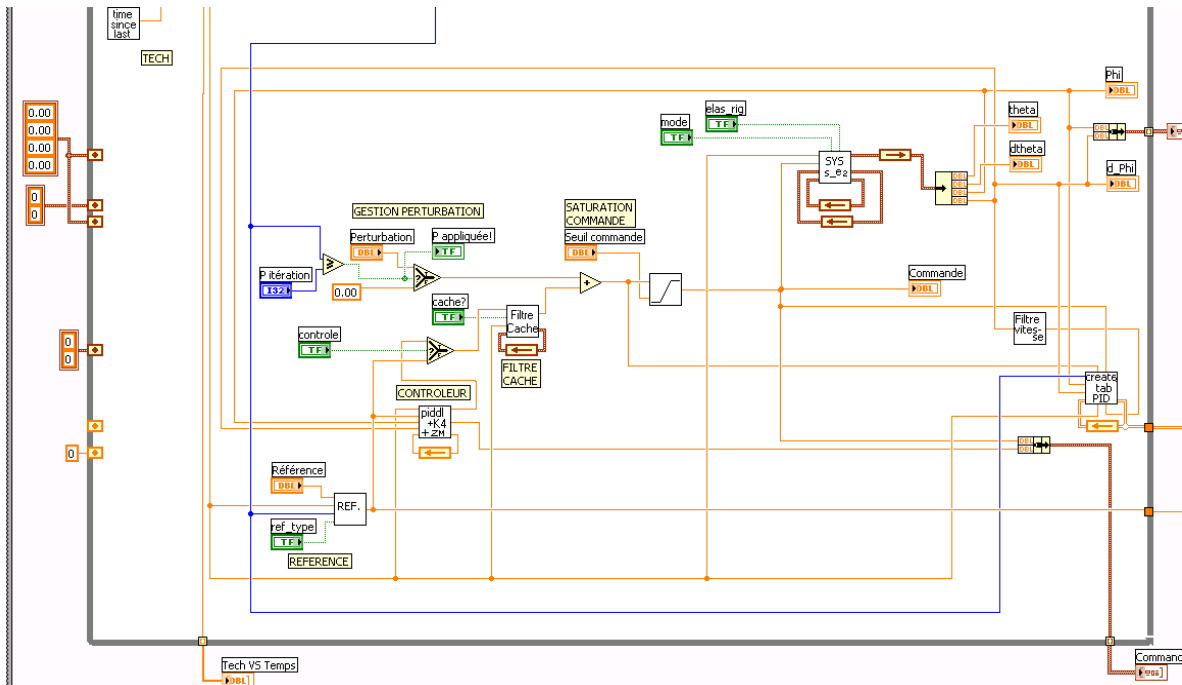


Figure 4: Diagram in LabVIEW (example from case study)

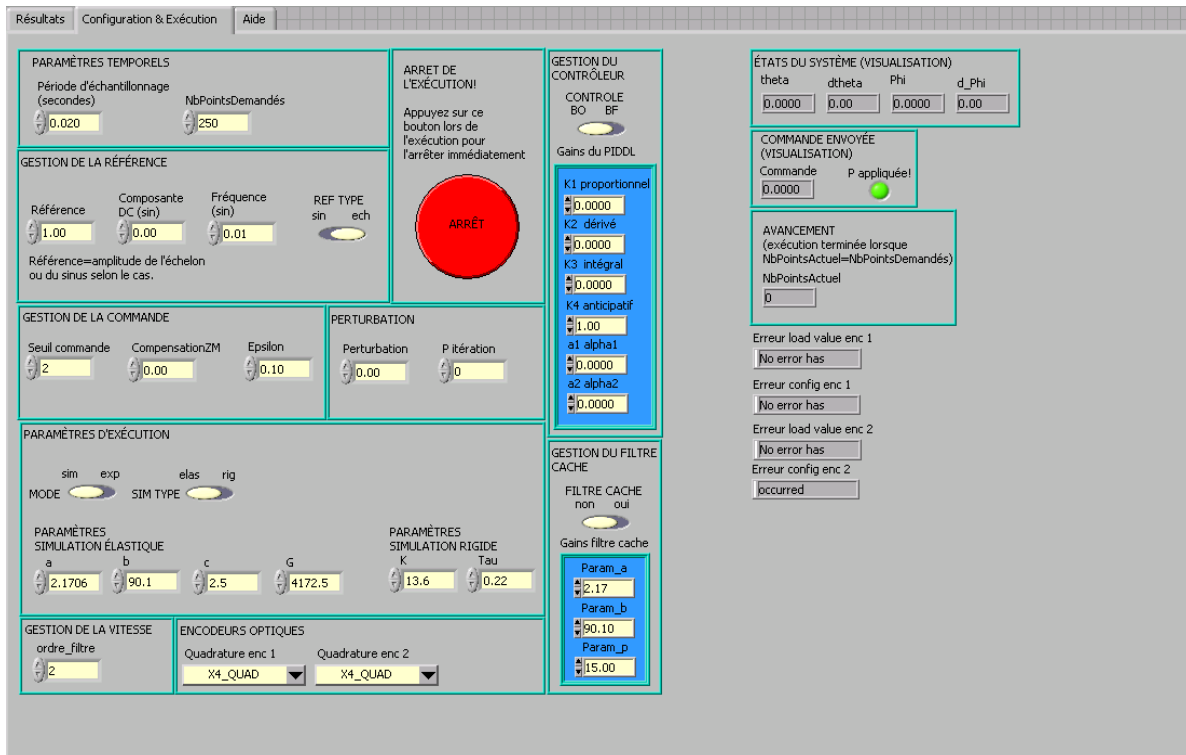


Figure 5: Panel in LabVIEW (example from case study)

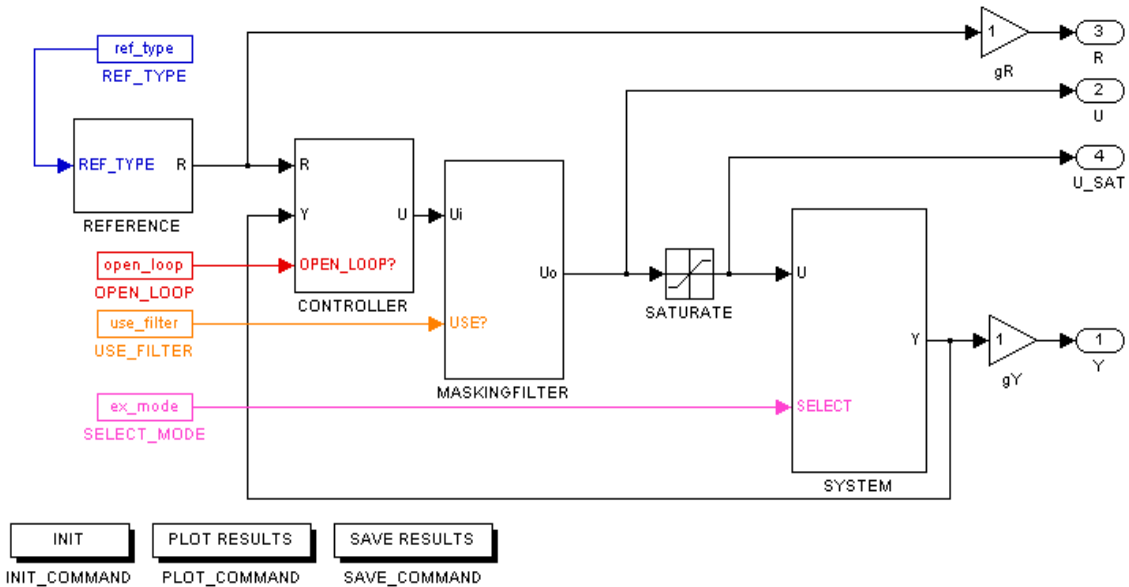


Figure 6: Simulation / real control model in Simulink (example from case study)

```

% test_bench_init.m
%
% This script initialize parameters values for the simulation of the
% test_bench model with Simulink / xPC Target. Once the model is compiled
% with Real-Time Workshop, every parameters need to be changed directly in
% the model (external mode connected) for the change to take effect on the
% target.

% model parameters
tech=0.001; stop_time=5;

% reference parameters
ref_type=1; % 1=constant; 2=sinusoidal
ref=2*pi; ref_freq=0;

% controller parameters
open_loop=1; % 0=closed loop control; 1=open loop control
k1=0; k2=0; k3=0; k4=0; a1=0; a2=0;

% masking filter parameters
use_filter=0; % 0=don't use; 1=use
a=2; b=60; p=15;

% system parameters (simulation parameters and execution mode)
ex_mode=1; % 1=real control; 2=rigid simulation; 3=elastic simulation
K=11.45; Tau=0.23;
a=2; b=60; c=3; G=3700;

```

Figure 7: Configuration script in Matlab (example from case study)

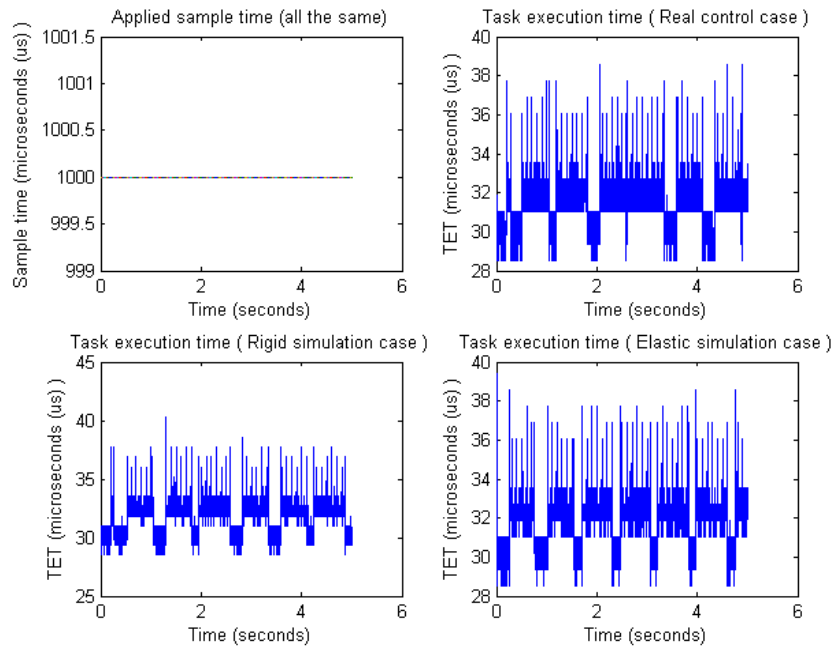


Figure 8: Sampling time results for performance testing

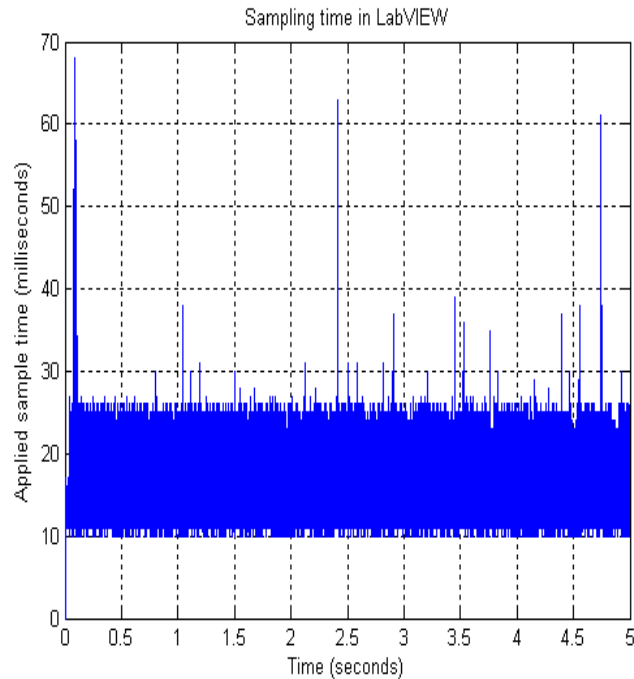


Figure 9: Applied sampling period in LabVIEW