Introduction to Real-time Control using LabVIEW^{TM} with an Application to Distance Learning*

Ch. SALZMANN, D. GILLET, and P. HUGUENIN
Swiss Federal Institute of Technology Lausanne, Switzerland. E-mail: christophe.salzmann:epfi.ch

This paper presents the approach taken to give engineering students the necessary competencies and facilities to implement real-time control solutions. This goal is achieved first by way of an introduction to the basic principles underlying real-time control. Then, by motivating the use of personal computers as a versatile alternative to more traditional implementation equipment. Finally, by combining LabVIEW and DAQ boards to form an integrated framework for fast prototyping of real-time control solutions. Control algorithms written in G (the graphical programming language of LabVIEW), in C or as S-functions (MATLAB scripts describing SIMULINK dynamical models) can be validated on laboratory-scale processes. The possible real-time control and monitoring of ongoing operations, either locally on the host computer or remotely via the Internet, is a key feature from an educational point of view. In fact, the chosen paradigm truly enable the 'learning by doing' approach. Moreover, this practical activity can be carried out at any time from anywhere to efficiently support automatic control study.

INTRODUCTION

FEEDBACK CONTROL LOOPS are implemented to increase dynamical performance or precision of scientific and industrial equipment. The basic principle of such loops is to take into account actual measurements in order to compute appropriate actuations that adjust the operational conditions to meet given requirements. Motion control and process control are two major application areas of this paradigm. Due to this broad application field and its interdisciplinary nature, Automatic Control is a fundamental subject usually taught in many engineering disciplines, such as electrical, mechanical and chemical engineering.

Practical experimentations are made during laboratory sessions where students can try out on real processes the material they learn during the class [1]. As a matter of fact, implementing a complete control solution from scratch requires knowledge not only of the matter studied but also of the different technologies needed to interface the real process, such as sensors and actuators, to the computer used to conduct the experiment. Computer knowledge such as hardware interfacing and real-time programming are also needed to carry out the experiment. Fundamentals of all these aspects should be taught to students in automatic control.

Acquisition of measurements and modification of actuations are the usual tasks carried out by LabVIEW^{TM} and DAQ boards. This implementation paradigm based on personal computer and standard operating systems constitutes a new trend in automation. It allows the user to avoid such aged, specialised or expensive solutions as analog PID loops, programmable logic controllers (PLC) or dedicated hardware based on digital signal processors (DSP).

To stress the advantage of the above-mentioned open paradigm, the students are provided with an integrated framework versatile enough to be quickly adapted to their different needs and backgrounds. This solution reduces the additional knowledge required to proceed with the implementation and helps students focus on essential concepts. To be attractive to the students, the framework needs to be highly interactive and offer a well-designed graphical user interface.

The recent increase in availability of personal computers at home and on campus has allowed students to exploit computer-aided instruction (CAI) tools to learn in their own way and at their own pace. Unfortunately, such independent work is not possible on laboratory-scale processes since these cannot be moved easily or duplicated in sufficient numbers. Therefore, experimental work is done in the laboratory according to a predefined schedule.

Both the recent developments in the Internet and the introduction of the World-Wide-Web (WWW) are opening the ways to interactive presentations, even remote access opportunities to real-world equipment. The availability and the capabilities of these new communication facilities, combined with the generalisation of computer use

* Accepted 9 September 1999.

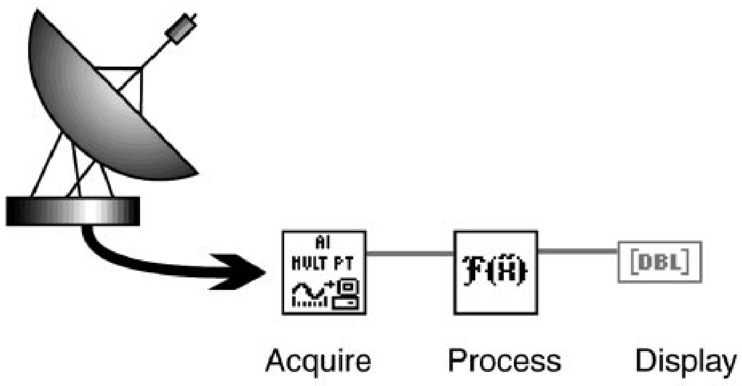


Fig. 1. Data acquisition and processing sequence. The AI_MULT_PT VI acquires a stream of data coming from the radar, these data are then processed (FFT) in one pass and displayed.

for data acquisition and control of real processes, enable the students to move from a presence at the equipment location to a more versatile tele-presence, thereby allowing remote experimentation.

This paper discusses the necessary competencies and facilities to implement real-time control solutions, either in traditional education or in distance learning. An example of ditributed applications using LabVIEW is given, where an electrical drive is introduced as convenient setup to study locally or remotely applications in motion control. This example serves as illustration throughout the paper. The advantages and the potential of local and remote experimentation supported by LabVIEW are expressed as conclusions.

FUNDAMENTALS OF REAL-TIME CONTROL

The notion of *real-time* implies that computer operations rely on absolute and irreversible time. These operations are generally performed accordingly with the evolution of a physical system. If necessary, the system-state is made available to the computer through appropriate interfacing devices, such as sensors and converters. The notion of *real-time control* reinforces the original definition by specifically emphasising actions carried out in addition to observations.

These notions can be illustrated by comparing data acquisition and processing (Fig. 1) with real-time control (Fig. 2). There are two main differences. In the former case, a finite number of acquisition cycles are performed to provide a stream of data for post processing operations. In the latter case, continuous operation cycles are performed. During these cycles only one data sample is acquired and immediately processed.

Moreover, in contrast to data acquisition, control operations require actuation values to be computed and written to output ports at every cycle. This removes the possibility of performing batch processing based on the data stored in an acquisition buffer (if there is any).

Appropriate actuators and amplifiers interfaced with the output ports enable the computer to drive

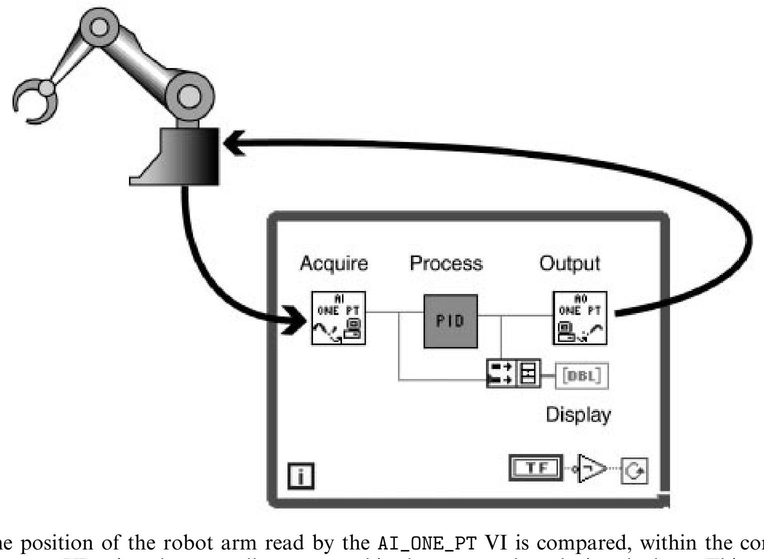


Fig. 2. Control cycle. The position of the robot arm read by the AI_ONE_PT VI is compared, within the controller (PID VI), to the reference value. The AO_ONE_PT VI writes the controller command in the output, thus closing the loop. This operation is performed at each loop iteration.

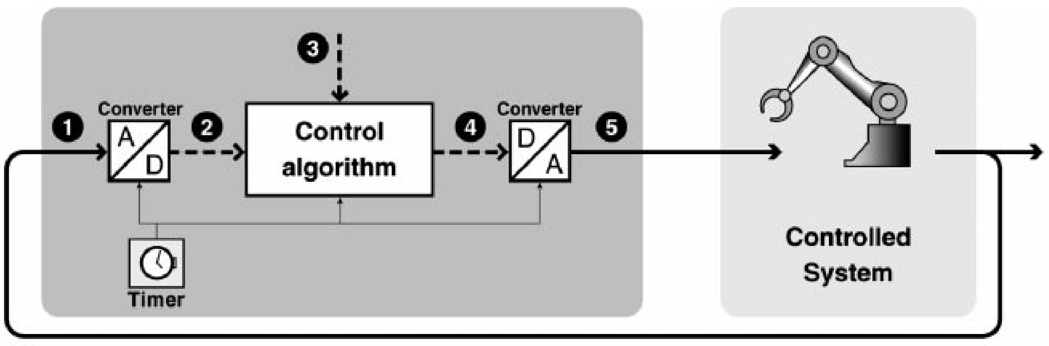


Fig. 3. Feedback structure. (1) The measurement is converted into a number (2) and compared to a reference value (3) within the controller. The resulting command (4) is converted to an analog signal (5) applied to the physical system.

the physical system with the necessary amount of energy.

Implementation constraints

Figure 3 presents the standard structure for a **feedback loop**. The output of the controller drives the input of a physical system, whereas the output of the physical system is reintroduced as the input for the controller, thus closing the loop. To guarantee an efficient and reliable control, some implementation constraints must be taken care of.

The operations performed by the controller have to comply with a standard **sequence**. The first operation performed is the measurement and the analog-to-digital conversion of the input signal (1). Once converted into a digital number (2), this value is compared to a reference value (3) resulting in an error signal which is used by the control algorithm to compute the command or output value (4). Other quantities may also be computed depending on the application (such as parameters or state estimates). The output value is then converted to an analog signal (5) and applied to the controlled system. This sequence of operations forms a **real-time task** (RTT), which is repeated at a fixed interval, called the **sampling period** or the **cycle time.** In advanced applications, the implementation of different real-time tasks with different sampling periods and priorities may be required. The sampling period must be adapted to the dynamics of the physical system. If the sampling period is too large, the control of the system will be lost and if it is too short, quantisation problems may occur.

To allow the use of the traditional control analysis and design techniques, the sampling periods are assumed to be constant. In other words, the **cadence** at which the real-time tasks are called must be as regular as possible. This is generally guaranteed by **interruptions**. Interruptions are generated either by an on-board timer (host computer) or by an external source located on the acquisition device (Fig. 4). These interruptions tell the operating system (OS) to switch from the current task to the real-time tasks according to their respective priority. Once the real-time tasks are completed, the OS resumes the original task.

The cadence is mainly limited by the underlying OS. Most of today's OSs only partially support real-time operations and therefore accurate cadence cannot always be guaranteed. The use of specialized OSs, dedicated processors or embedded systems are required for mission-critical implementations where, for example, people or equipment safety must be guaranteed.

The accumulated duration of the real-time operations should be smaller than the sampling period for obvious roll-back reasons.

The time between the occurrence of the interruption and the launch of the real-time tasks is called the **latency**. This time is variable and OS dependent but should be much smaller than the sampling period.

Under tight time constraints, the real-time tasks (RTT) are generally monitored by a **supervision task** (ST). The supervision task has a lower priority than RTT and runs asynchronously. The ST

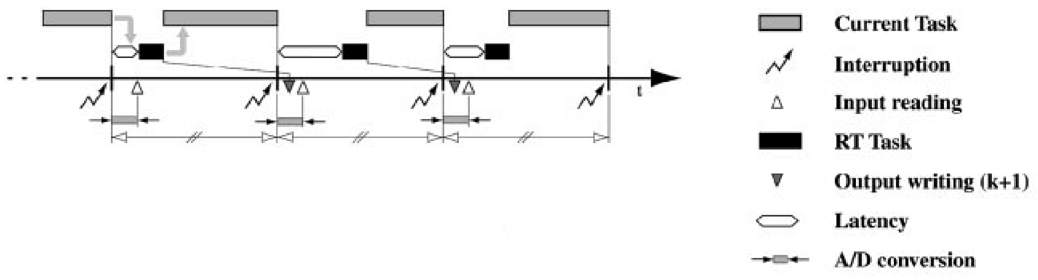


Fig. 4. Interruption cycle. The Interruption triggers the signal acquisition and tells the OS to switch from the current task to the RT task. The time between the interruption and the execution of the RT tasks is called the latency.

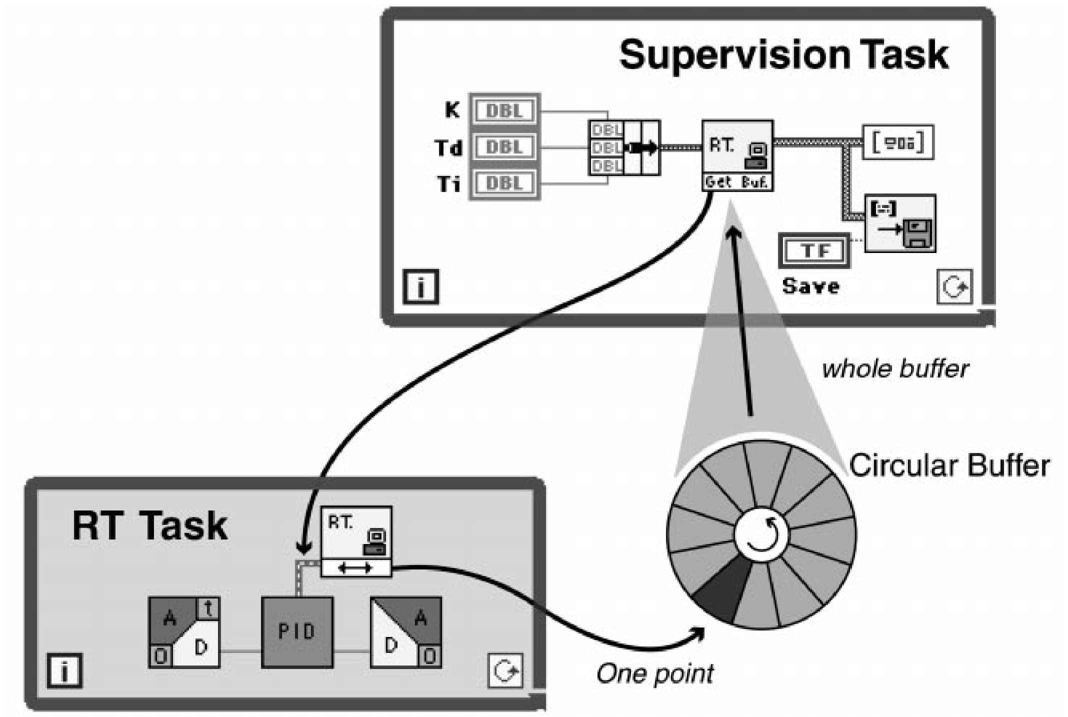


Fig. 5. Data sharing principle. The RT task loop reads the input, executes the control algorithm (PID), writes the output and saves one position in the circular buffer at each sampling period. The supervision task uploads the whole buffer in one operation and displays its content. It also updates the RT task parameters.

performs all the functions that are not time dependent such as managing the user interface. The ST exchanges data with RTTs through a circular buffer (Fig. 5). The RTTs update one position of the buffer at each interruption, on the other hand the ST uploads the all buffer in one operation. The cycle time of the supervision task is generally much longer than the cycle time of RTT.

When the user wants to modify a RTT parameter, the RTK must reflect this modification in real-time (interactivity). The user parameters are transmitted asynchronously by the ST to the RTT through a **buffer.** This asynchronous buffer is not updated at each interruption, but only when a parameter is modified.

Requirement for real-time control in education

The real-time control capabilities required in education are generally reduced compared to industrial applications, mainly because laboratory experiments only show one aspect of the theory at a time. Usually, only one physical system is controlled and only one real-time task is sufficient, such as open-loop measurement, identification or closed-loop control.

To minimize the time spent by the students in the laboratory, the application should be well designed and well integrated within the environment. The user interface should be of high quality. This might be different from industrial products where the GUI is less important in the exploitation phase. As a consequence, enough processor time should be reserved for the supervision task which allows interaction between the user and the running controller.

The classical trial and error learning approach implies that successive changes should be made in the control algorithm before getting the required or desired dynamical performance. This can be developed using a fast prototyping environment where the design-implementation-test cycle should be as short and as integrated as possible.

The use of a software-only solution guarantees a low price and a great expendability. Moreover, remote experimentation, presented later, relies on a software-only solution for the distant users.

FUNDAMENTALS OF PID CONTROLLER

Typical controllers are implemented as numerical algorithms that are designed to compute, at every sampling time $tk = kh$ (where h is the sampling period and k is the sample index), a correction factor $u(k)$ which is added to the nominal excitation of the controlled process. The nominal excitation is a predefined signal based on a priori knowledge and issues to bring the process to desired operation conditions or to track given trajectories. The correction is necessary to both handle unexpected disturbances which affect the dynamic behaviour of the controlled process and to cope with the unavoidable imprecision of the nominal excitation.

The most commonly used controller is called the PID. The corresponding algorithm issues a

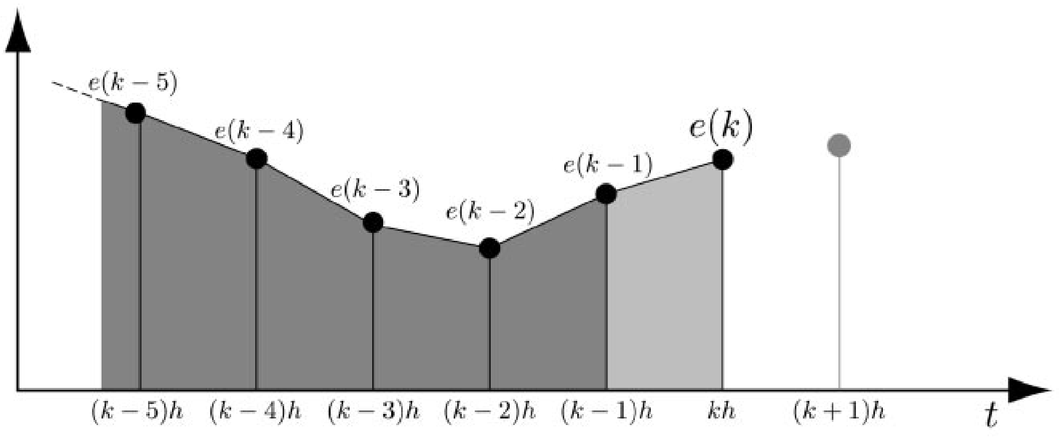


Fig. 6. Time evolution of the error used by the PID controller.

correction according with the time evolution of the actual error $e(k)$ existing between the desired and the measured output of the controlled process, respectively denoted $y_c(k)$ and $y(k)$. The acronym PID is due to the way the correction is computed. The first term $u_P(k)$ of the correction is **proportional** (P) with the current error:

$$u_P(k) = K_P e(k) \tag{1}$$

The second term $u_I(k)$ takes into account the error history in order to cancel out an eventual bias. To avoid the storage of all previous samples, the past behaviour is summarised using the **integral** (I) of the error signal. In the discrete case, the integral can be evaluated in various ways. The method chosen is the sum of the trapezoidal areas obtained by linking the successive error values (Fig. 6).

$$u_P(k) = u_I(k-1) + K_P \frac{1}{T_I} \left[\frac{e(k) + e(k-1)}{2}\right] h \tag{2}$$

In addition, the correction has to take into account the speed of the error evolution. If the error varies quickly, the correction has to be stronger to avoid overshoots. Such an action is obtained by introducing a last term $U_D(k)$ which is proportional to the **derivative** (D) of the error signal computed using the last two available samples.

$$u_D(k) = K_P T_D \frac{e(k) - e(k-1)}{h} \tag{3}$$

Finally,

$$u(k) = u_P(k) + u_I(k) + u_D(k) \tag{4}$$

In equations 1, 2 and 3, the factors K_P, T_I and T_D are the tuning parameters defining the behaviour of the closed-loop system.

It is essential to underline that the PID described in this chapter is purely discrete, which is the right form for a computer implementation. Rigorous techniques based for example on the Z-transform can be applied for analysis and design purposes [2]. However, it is often sufficient to carry out a single open-loop experiment on the physical system to be controlled to get a convenient set of parameters. The idea is to apply a step of level U as an excitation signal of the system (Fig. 7). The corresponding variation of the output provides the slope a at the inflection point (or at infinity if the response reaches no stationary value) and the time

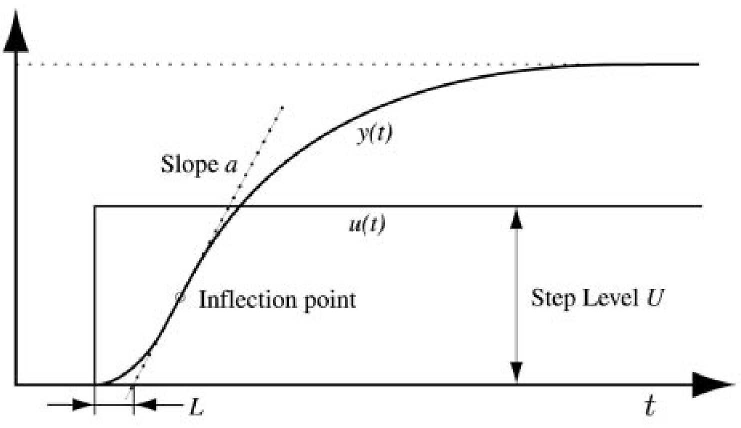


Fig. 7. Step response of the system to be controlled.

Table 1. Parameters of the P, PI or PID controller.

Controller	K_P	T_I	T_D
P	$\frac{U}{aL}$	—	—
PI	$0.9\frac{U}{aL}$	3.3 L	—
PID	$1.2\frac{U}{aL}$	2 L	0.5 L

interval L between the step and the crossing of the tangent at the inflection point with the time axis.

The convenient parameters are listed in Table 1, with respect to the results of the experiment and the desired controller type.

The resulting sequence of operations which have to be implemented to complete the PID correction at time k is:

- acquire the measurement $y(k)$;
- get from the user the current PID parameters K_P, T_I and T_D, as well as the reference signal $y_c(k)$;
- compute the error $e(k) = y_c(k) - y(k)$ (some filtering can also be considered here);
- compute $u_P(k)$, $u_I(k)$, and $u_D(k)$ if applicable (the states of the controller $u_I(k-1)$ and $e(k-1)$ have to be available);
- compute the sum $u(k)$ and limit its value in a range compatible with the physical devices (power amplifier, motor);
- output the previously computed control signal $u(k)$ (or wait for the next sampling period to do so).

PID CONTROLLER IMPLEMENTATIONS WITHIN LabVIEW

This section presents the different alternatives for the real-time implementation of a PID controller using LabVIEW (National Instruments propose a PID toolkit with autotuning). The first solution is a plain LabVIEW implementation written in G. The next two solutions combine LabVIEW with a real-time kernel allowing the code of the control task to be written in C or as a MATLAB/Simulink S-Function. The possibility to call a C routine or an S-Function enables the use of legacy code. The last implementation relies on the new LabVIEW RT. The professional system Concurrent PowerMAX as well as other embedded solutions are not evaluated in our review.

Plain LabVIEW

In LabVIEW, the execution of the program, called the Virtual Instrument (VI), is paced by the data flow. This flow can be slowed down or interrupted by some external events such as network accesses, mouse movements, disk operations or other OS events. The cycle time for a control loop written in LabVIEW is therefore non-deterministic and dependent on the time used by the other VIs running and to the computer's other activities. This implies that control of the physical process might be lost when these events occur (Fig. 8).

To guarantee the correct behaviour of the system, the user must ensure that the duration of these events is much smaller than the chosen sampling period. This can be partially alleviated by using the multi-threaded functionality of LabVIEW and by setting appropriate priorities to the different VIs. The main application is split into Sub-VIs, each representing a different thread (Fig. 9).

Higher priority is given to the PID code, which can be written using the regular LabVIEW functions or by using a formula node. The lowest priority is given to the user interface update which corresponds to the supervision task.

The appropriate use of the DAQ board can also compensate for the varying latency since many of

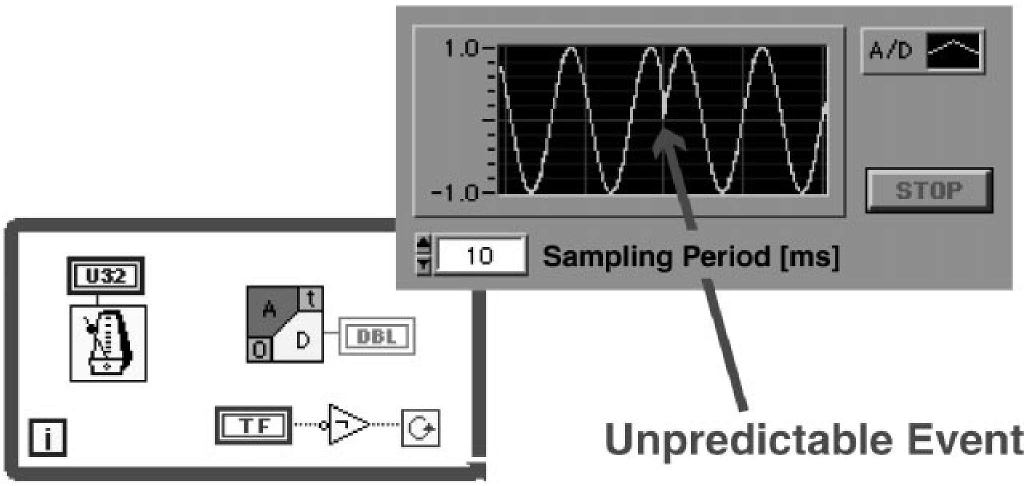


Fig. 8. Unpredictable event occurrence. The LabVIEW execution flow can be interrupted by some external events such as network accesses, mouse movements, disk operations or other OS events. The cycle time for a control loop written in LabVIEW is therefore non-deterministic.

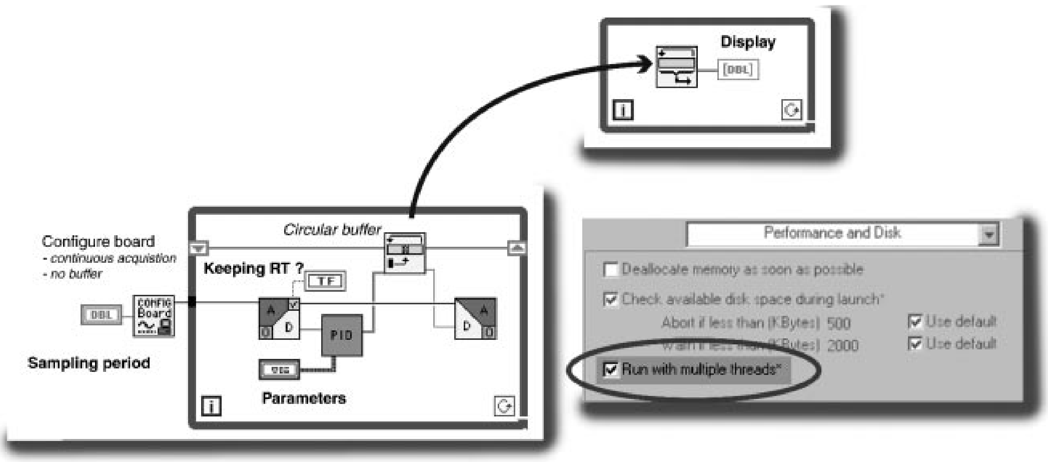


Fig. 9. Multi-threaded implementation. The display loop and the real-time (RT) loop run in different threads. The Display loop has a lower priority than the RT loop. The `CONFIG_BOARD` VI sets up the DAQ board for continuous acquisition without buffer. The on-board timer insures the accurate sampling period. The user is informed of a faulty operation (i.e. more than one sample in the buffer meaning that the loop was slowed down or interrupted) by the `Keeping_RT_?` button.

today's universal acquisition boards have the required timers and circuits to generate interrupts. The advantage of using the DAQ board interrupts is that such signals can also trigger, very accurately, the acquisition and the buffering of the data, this independently of other computer activities (Fig. 9). In such a way, the delay separating the interrupt issuing and the sampling is inexistent. This is essential to achieve a predictable behaviour (invariant sampling period) and to conform with the applicability conditions of the stability criteria. In this configuration, the latency corresponds to the delay between the interrupt occurrence and the effective buffer retrieval by the `Read AD` sub-VI. Fortunately, this delay is not critical to the control loop reliability as long as it does not exceed the sampling period. If this happens, one or more samples are not taken into account, which can be disastrous. To detect such an event, the `Read AD` sub-VI must ensure that there is one and only one sample in the acquisition board buffer. This means that the next interrupt has not occurred and that the LabVIEW loop cycle is smaller than the sampling period (as expected). In this case, the main loop is running as fast as possible and waits for a measurement to arrive in the board buffer.

Alternatively, without the use of the DAQ board timers, the LabVIEW main loop should cadence the execution of the controller using the `Wait Until Next ms Multiple` VI (Fig. 10). In this case, the AD conversion is triggered at each execution of the main loop and the controller has to wait for the data to become available. Since LabVIEW execution can be interrupted or delayed, it is not guaranteed that the given cadence can be followed. Moreover, undesirable jitter in the sampling time is impossible to avoid.

This jitter of the sampling period is particularly critical for the computation of the derivative term in which h appears explicitly. If the desired value of h is used instead of the real one, huge errors may occur. To compensate for the sampling period variations, a timestamp needs to be stored with each measurement sample. This timestamp will be used to determine the real sampling period to be use in the numerical derivation, as well as for the display and the storage of the data.

LabVIEW with the real-time kernel

To improve the performance of controllers running within LabVIEW and to reuse the legacy code (C and S-Function) which may have been developed and tuned for other applications, a real-time kernel (RTK) which handles all the real-time operations has been designed, thus removing the need for LabVIEW to do this [3]. The RTK is able to execute the user's real-time task (RTT) repetitively at a fast and accurate pace. To achieve these requirements, the RTK relies on interrupts.

The real-time kernel is implemented in C as a code interface node (CIN, LabVIEW external code). This kernel allows one or more user routines, typically a controller routine, to be called by the OS at interrupt time.

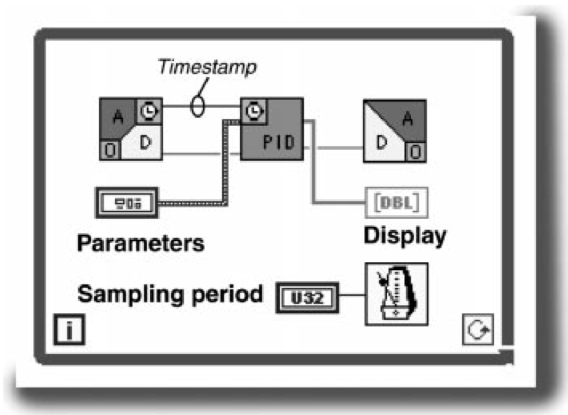


Fig. 10. '`Wait Until Next ms Multiple`' loop cadencing. The AD VI adds a Timestamp to the measurement to compensate the sampling period variation.

The RTK should be seen as a standalone background application, which communicates through shared memory with the foreground application (LabVIEW) holding the user interface. The RTK has two main functions: the first is the communication with the physical process via acquisition boards. The second is the management of the user-defined RTT, which can be any type of real-time operations such as real-time control, real-time simulation or for example on-line identification.

The user-defined real-time tasks are called by the RTK at each sampling period or at a multiple of the sampling period (sub-sampling). If there is more than one RTT defined for the same board, the RTK calls them in a pseudo-parallel fashion. This simplifies the implementation of parallel or complex operations. In the example of a single RTT performing a cascade control with two PIDs, the RTT can be split into two single PID controllers. In the same manner, sub-sampling allows two different tasks such as filtering and control to be called at different paces. Instead of having the controller called at a faster than needed sampling rate in order to filter the input signal(s), these two operations are implemented in separate routines: the filtering operation which is performed at each sampling period, and the control operation which is called, for example, every twenty interrupts. The filtered signal is shared between the filter and the controller using the buffer. This method lowers the processor's load and makes the RTT easier to write.

The input and output data defined by the user are stored in an internal buffer. This buffer is updated synchronously with each interrupt. When called by an interrupt, the RTK puts the newly acquired values into the buffer and then retrieves the value for the next outputs from the same buffer. For display purposes, this buffer can be partly or completely retrieved asynchronously by LabVIEW. Besides input and output data, other information such as timestamps, execution errors, and virtual channels holding internal states of a controller (for example the previous integral values) are stored in the buffer.

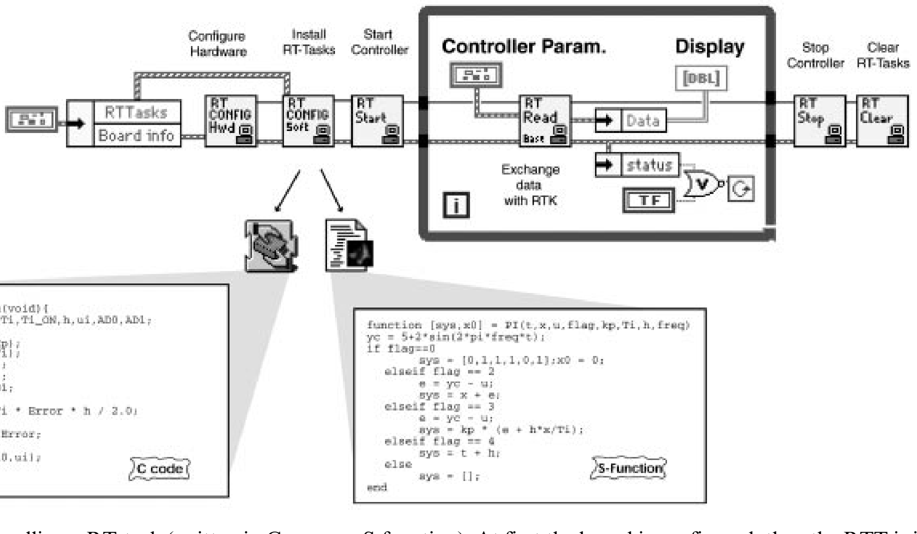
A set of VIs allows to control all the required operations related with the RTK such as configuring the hardware, configuring the RTTs and starting and stopping controller routine (Fig. 11). Another VI performs the data exchange from and to the controller routine. Those VIs are very similar to the current DAQ VIs.

The RTK allows the user to install one or more real-time tasks. These tasks are Dynamic Linked Libraries (DLLs written in C) or S-Function (written in MATLAB format). To avoid having to worry about complex compiler environment issues, the compiler can be controlled directly by a Real-Time Framework [3]. Currently, only the MacOS platform is supported. A freeware subset of this environment is available on-line (http://iawww.epfl.ch).

MATLAB and Simulink (http://www.mathworks.com) are widely used in the engineering world. The simulation and the design of controllers can be conducted using the MATLAB/Simulink language. Instead of rewriting the code developed in C or in G, the RTK can reuse the legacy code by calling, at interrupt time, an interpreter, which implements a large subset of the MATLAB syntax. The built-in interpreter has been written with the primary purpose to be run in real-time. Due to the extensive computer load resulting from the interpretation which is carried out at every sampling period, the achievable cycle time is larger (by a factor of 10) in that case than in the one when calling C code.

LabVIEW RT

National Instruments proposes an intelligent DAQ board containing all of the necessary components to develop real-time systems. The board



```
long UserFunction(void){
double Error, Kp,Ti,Ti_ON,h,ui,AD0,AD1;
long   err;
 err = Get(0,1,&Kp);
 err = Get(0,4,&Ti);
 err = AD(0,&AD0);
 err = AD(1,&AD1);
 Error = AD0 - AD1;

 ui = ui + Kp / Ti * Error * h / 2.0;

 Cmd = ui + Kp * Error;
 err = DA(0,Cmd);
 SetInternalBin(10,ui);
 return err;
}
```

```
function [sys,x0] = PI(t,x,u,flag,kp,Ti,h,freq)
yc = 5+2*sin(2*pi*freq*t);
if flag==0
        sys = [0,1,1,1,0,1];x0 = 0;
   elseif flag == 2
        e = yc - u;
        sys = x + e;
   elseif flag == 3
        e = yc - u;
        sys = kp * (e + h*x/Ti);
   elseif flag == 4
        sys = t + h;
   else
        sys = [];
end
```

Fig. 11. The RT-kernel calling a RT-task (written in C or as an S-function). At first the board is configured, then the RTT is initialised and started. During the loop execution data (parameters and buffer) are exchanged with the RTK via the `RT_READ` VI. Upon termination, the RTT is stopped and removed from the memory.

consists of two components: a processor board and a DAQ daughter card. The processor board includes many of the PC components, but does not contain hard-drive or other standard I/O devices. The daughter card is similar to other NI DAQ products.

A multi-threaded real-time engine (RTE) runs on the processor board using a real-time operating system (RTOS). The RTOS ensures that the scheduler and other operating system services comply with the real-time requirements.

The RTE executes LabVIEW RT programs which are exactly the same as the standard LabVIEW programs but with some limitations. For example, there are no disk access and no Ethernet communication capabilities due to the lack of peripherals on the RT board and due to the potential RTE performance degradation they would imply. Attempting to use these unsupported functions in an embedded LabVIEW RT application produces standard LabVIEW error codes.

The development system runs on Windows (NT/9×). Before being executed, the VIs are downloaded on the RT board. Once on the RTE, the VIs run without any outside interference thus allowing deterministic cycle times. The VIs running on the RTE will still be running even if the host computer reboots. For display and other purposes, the VIs on the RTE exchange data with the RT development system via messages or shared memory. The display of the user interface is performed automatically without any user programming.

There are no programming differences between LabVIEW RT and the standard version of LabVIEW. Having the critical part running on the dedicated RTE ensures that the embedded code will not be interrupted by any other outside event. The PID controller can be cadenced with the **metronome** (`Wait Until Next ms Multiple`) function (Fig. 12). This function puts the current thread into sleep and allows other threads to run. By giving the **time critical priority** to the embedded PID VI, it is ensured that the loop execution time is guaranteed, this provided that the loop time is smaller than the sampling period. If the time critical thread consumes most of the processor time, the others threads, such as the communication thread, which exchange data with the RT development system will not have time to execute resulting in a sluggish user interface update. The user interface may even look frozen if the time critical thread consumes all of the processor time. In that case, the embedded VI is still running, but has no time to transfer the data to the development system. This might be suitable for industrial systems not requiring a user interface, but for education it is important to interact in real-time with the controller. Thus, the sampling period should allow enough time for the user interface management.

Concurrent PowerMAX real-time LabVIEW

National Instruments has ported LabVIEW to the Concurrent PowerMax real-time architecture. Concurrent computers develop a real-time

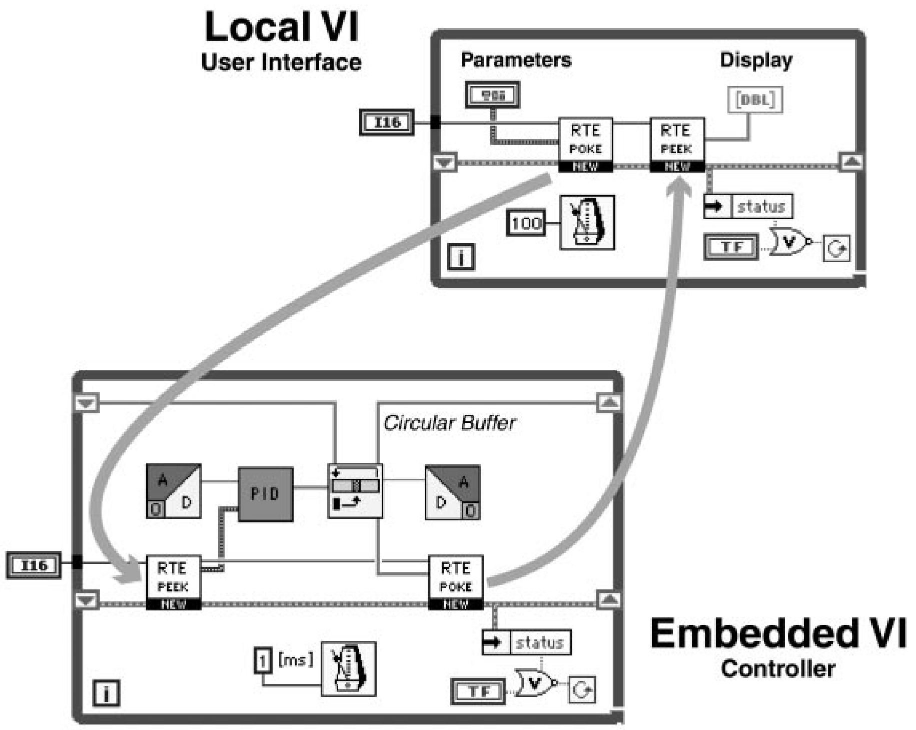


Fig. 12. The local and the embedded VI. The emdedded VI containing the controller code (PID) is downloaded to the LV-RT board. It exchanges data (parameters and circular buffer) with the local VI containing the user interface by using the `RTE_PEEK` and `RTE_POKE` VIs.

Table 2. PID implementation solutions.

Solution	Coding	Cycle time	Platform	Interrupt	Hardware
Plain LV	G	10 ms[2]	All	soft	DAQ
Multi-threaded LV	G	5 ms[2]	All[3]	soft	DAQ
RT Kernel calling C	G[1] & C	0.2 ms[2]	Mac[4]	soft/hard	DAQ
RT Kernel calling S-Function	G[1] & Matlab	5 ms[2]	Mac[4]	soft/hard	DAQ
LabVIEW RT	G	1 ms	Embedded on PCI, PXI	hard	RT[5] & DAQ
PowerMax	G	not tested	PowerMax	hard	PM[6] & DAQ

[1] LabVIEW is used for the supervision task (user interface).
[2] Depends on processor performances.
[3] Mac version is currently not multi-threaded.
[4] Porting the RTK to the PC platform is under evaluation.
[5] Require the embedded LV board.
[6] Require a PowerMax computer.

multitasking UNIX-based operating system which can guarantee a given latency and a deterministic real-time response. This is a scalable, professional (read expensive for education) system and therefore outside the scope of this paper.

Discussion

Different solutions have been presented for implementing a PID controller within LabVIEW. Depending upon the financial and technical requirements, different solutions can be selected. Table 2 summarises the different possibilities.

In education, it is important to consider the coding simplicity according to the students background which may vary among engineering majors. The acquisition and maintenance cost of the real-time environment are also important. The part of the budget dedicated to hardware should be kept as small as possible, limiting the possibility to use professional hardware solutions. A software-only solution should be chosen. Moreover, due to the strong interaction between the hardware drivers, the OS and the control software, real-time environment upgrades have to be carefully planned.

As it will be shown in the next chapter, a solution without additional hardware is preferred since distant users do not want to duplicate the DAQ board or the embedded real-time environment needed to locally control the remote system.

EXTENSION TO DISTANCE LEARNING

Motivation for remote experimentation

Currently, traditional and virtual universities propose on-line courses based on electronic documents and multimedia presentations enriched with video and audio broadcasting [4]. While the students can take these classes from a remote location they still have to come onto the campus for the laboratory practice. This restriction can be overcome by allowing students to access the laboratory facilities from a distant location to carry out hands-on sessions [5].

The development of remote-experimentation facilities is also motivated by the fact that the demand for access to laboratory facilities is growing rapidly in all engineering colleges. At the same time, the number of students is increasing while the allocated laboratory resources do not keep pace with this change. Being able to make the laboratory infrastructure accessible as virtual laboratories, available 24 hours a day and 7 days a week, goes a long way towards addressing these difficulties, and would also contribute to lowering the costs of operating the laboratory in the long term. Moreover, students are able to carry out experimentation at the precise stage in their learning process when they need to compare their knowledge to reality. This is an additional benefit of such a new paradigm introduced in distance learning. The increased availability is obtained by allowing students to reach the laboratory facilities via the Internet using a modem, or from other points of network access, such as computers available at different campus locations. The Internet offers many advantages over other technologies, which makes it the medium of choice. The first advantage is its price and availability. Nowadays, almost every household has a telephone line which makes a potential Internet connection. Current technologies such as ISDN or 56K modems give enough capacity to transmit voice and video with an acceptable quality. In a remote experimentation, not only the users may be distributed all over the world, the experiments can also be distributed among the potential users (such as universities) thus giving an opportunity to reduce the costs associated with laboratory facilities by sharing unique or expensive equipment.

The learning environment is based on a client/server architecture (Fig. 13). Given its fully computer-based implementation, the laboratory environment can easily be expanded for remote manipulation. The main concept in turning the locally-controlled setup into a remotely-controlled one consists in moving the user interface away from the experiment. Two distinctive parts result: the remote client and the local server.

- The **remote client** is a computer equipped with the functionality necessary to observe and to act on the remote experiment. The client application is a VI compiled for the target platforms. This VI provides the user with a complete interface to

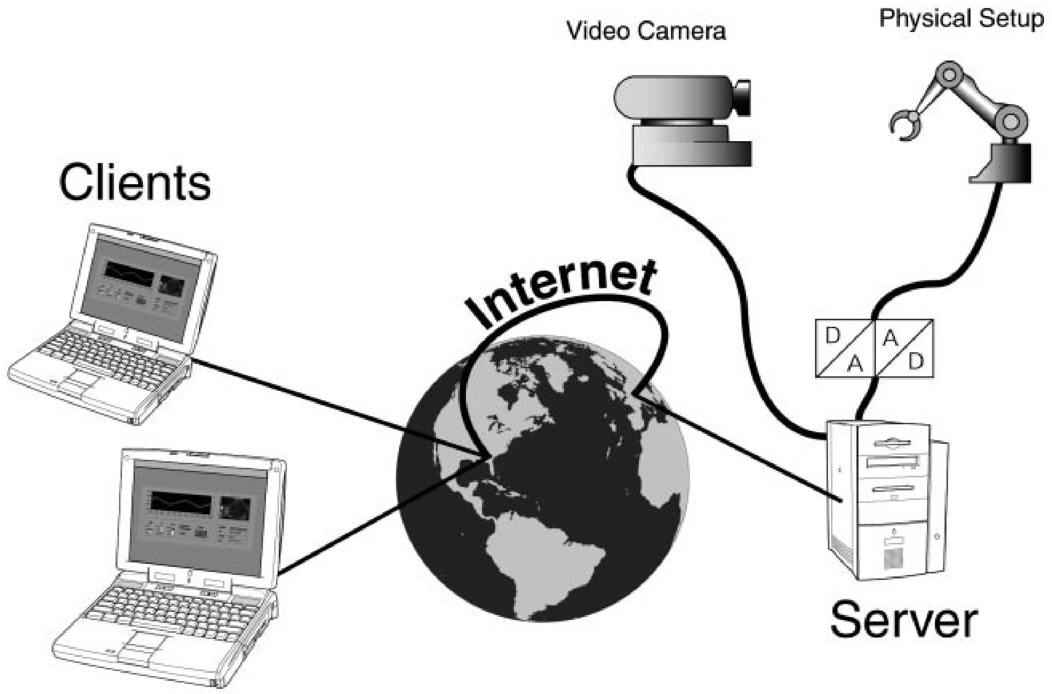


Fig. 13. Architecture for carrying out real experiments in distance learning.

the real process. It is used to generate excitation signals and observe corresponding responses. The main concept of such an interface is to provide a general view of the physical process evolutions, and to allow full control of the operations.

- The **local server** is the computer located near the real process and equipped with the hardware interface to the sensors and actuators. The video camera and microphone can be assimilated to sensors. The server application receives the client commands and transmits them to the real process. It also returns the states of the physical process to the client including an image.

Three modules are necessary to build the client and the server applications: the GUI module, the RT module and the COM module (Fig. 14). The application used locally can be split in two modules: the real-time controller (RT) and the user interface (GUI). The client and the server application can be designed by adding a communication module to the two existing modules. The client application is made up of the GUI and the communication modules. The server application is made up of the controller and the communication module. The server may require a basic user interface for supervision of the ongoing operations. The communication module allows the client and server applications to exchange information with other computers distributed in different geographical locations. This module also takes care of security issues regarding network management. For example, it prevents unauthorised access and schedules logins to avoid conflict.

By isolating carefully these modules in the development process, it is easy to port a local solution to a remote one, or port the remote solution to different physical systems.

Requirements

Distancing the user from the local experiment while keeping the same amount of benefits as local experimentation is challenging. Not only the same degree of interactivity must be maintained, but the remote solution must also allow the user to 'feel' the real experiment. During local experimentation students can use their senses of vision and hearing to perceive the effect of their acts on the control system. In a remote experimentation mode, this specification is addressed by providing audio and video feedback information in addition to the information given to the remote computer through the graphical user interface (GUI). Obviously, such feedback needs to be given in a reasonable amount of time, minimising the misleading (and most likely also disturbing) effects of signal-transport delays. For example, a remote user will not find acceptable feedback that arrives 30 seconds after an action has been accomplished, while the local response is achieved in fractions of a second. Consequently, fast system responsiveness is a key goal in all developments for remote experimentation. As might be expected, ideal instantaneous responses are not possible, but response times should still be minimised.

In addition to the need to remotely 'feel' the physical system, it is also necessary to remotely 'touch' it. For example, perturbing the system by hand to observe the reaction of the controller should be possible. This is achieved by introducing an additional actuator or by artificially altering

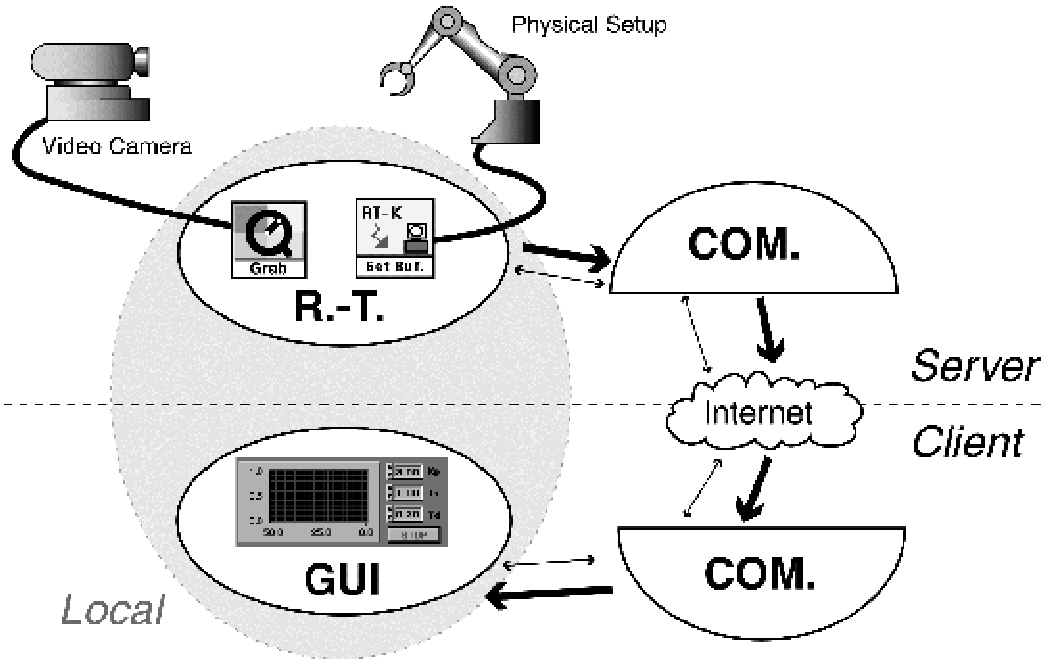


Fig. 14. The three modules for building the client and server application. The RT module handles the real time operations (video acquisition and physical setup control). The COM. module located on both ends transfers the data intelligently between the client and the server. The GUI module displays the incoming data and sends the parameter changes to the server.

some control or measurement signals to mimic the desired effect.

The interactivity must be adequate, otherwise the gain induced by remote experimentation will be minor and the facility will probably not be used. The objectives for a fast application such as motion control, where the user should be able to catch the dynamics of the process, are different from the objectives of a slow chemical plant where the user needs mostly to deal with the complexity of the system. The solution presented here focuses on fast dynamic systems such as electromechanical processes. Since they have mobile parts, they can be monitored remotely by cameras. In the case of visually static systems such as thermal systems, remote observation can be enabled by the use of sensors that modify their appearance according to the measured state.

Streams and adaptation

Adaptation to the network (i.e. Internet) load is necessary for using wisely the available bandwidth. In the future, unfriendly applications, which do not adapt might be banned from networks if they do not behave adequately.

Different kinds of information are exchanged between the server and the clients according to four different classes:

- the data stream representing the measurements made on the physical system;
- the audio/video stream acquired by the camera;
- the parameter stream reflects the user actions on the client side;
- the administrative stream which deals with the login/logout issues [6].

The adaptation is made by assigning a different priority to each stream. An algorithm, taken the network load seen at the client side into account, determines the server bandwidth usage. Based on stream priorities and the information coming from the client, the server optimises the packet size and packet rate. The server application can adapt the amount of information transmitted by increasing the image compression factor and/or by decimating the measurements. This scheme can adapted in real-time to a wide range of bandwidth from modem line to LAN connection. A specific technique is used to recover packets, which are lost during the transmission.

DISTRIBUTED APPLICATIONS USING LabVIEW

Different solutions exist for controlling a computer remotely. Maybe the simplest one is the sharing of the remote screen and the redirection of the different local input devices such as the mouse and the keyboard. The information exchanged by the **screen sharing application**, such as Timbuktu (http://www.netopia.com), is mainly pixels representing the remote screen. The data used by the remote experimentation have a more compact representation and they can be updated/transmitted more efficiently since conventions exist between the client and the server. For example when a new point is added to the measurement display (1 in Fig. 18), the screen sharing application will update and transmit the all display to the remote computer. A better alternative is to only transmit the new point, resulting in a much more efficient use of the bandwidth.

National Instruments provides a fully featured **web server** written in LabVIEW. When running this server, the front panel (FP) of running VIs can be transmitted to a Web browser and updated at regular intervals using the server-push/client-pop technique. The server also support CGIs (common gate interface) written in LabVIEW such as image map. The combination of different VIs (FP image, CGI, and Form) can provide the client with a view of the remote setup. The user will be able to interact with the server within the limitation of current Web technology, i.e. the slow image update (a few frames per minute), the limited use of the *form* format to transmit information to the server and the rather cumbersome LabVIEW CGI programming required on the server side.

Nacimiento (http://www.Nacimiento.com) software proposes AppletVIEW, a toolkit which provides users with a complete development environment for creating Java pallets as front-end instrumentation panels that communicate with a LabVIEW server. Web pages may contain knobs, sliders, switches, and charts that are actual controls that communicate with the AppletVIEW VI. This is done without any Java programming.

LabVIEW 5 introduced a new mechanism, called VI Server, to programmaticaly access LabVIEW objects and functionalities. The server relies on TCP to exchange data between VIs (local and remote). This is done transparently and no specific knowledge is required. The security is defined on the server side. Different methods can be invoked, for example a VI can remotely set a control value, open a given VI or print the front panel. Another possibility is to call a remote sub-VI by invoking a **call by reference node.** The principle is similar to a remote procedure call (RPC) under UNIX. The only difference between a local or a remote call is the need to define the IP address and IP port of the remote machine. Prior to calling the sub-VI, the user needs to establish a connection with the server. On termination, the connection needs to be closed. A simple example of a remote machine transmitting its local time using the call by reference node is shown in Fig. 15.

Four ‘call by reference VIs’ were used to remotely control the physical setup. The first VI transmits the controller parameters from the client to the server, the second VI sends the measured values from the server to the client, and the third and fourth VIs implement watchdogs to detect if the client or the server are still available. This method has the advantage of being easy to implement and the performances are good on a lightly loaded local area networks (LANs). On the other hand, this method suffers from the TCP limitation (TCP slow start) when the network load increases or when the connection is not reliable implying packet losses.

LabVIEW UDP tools are used to overcome the TCP limitation, they allow full control of the bandwidth usage. This method is well suited for transmission over the Internet. UDP does not suffer from the slow start limitation, but this has a price: the packet delivery is not guaranteed. In other words UDP is connectionless, which means that the server has no knowledge of the packet arrival at the client side. The client application has the responsibility to inform, if required, that a packet has been lost and ask for its retransmission (this is done automatically under TCP).

For some applications, the packets can be lost without affecting the client application. For example, when transmitting a movie, a frame can be lost and the movie is still understandable. Remote experimentation belongs to this category and implements an advanced control scheme on top of UDP.

Figure 16 presents a local time server using UDP. It has the same functionalities as the previous example presented. The client application listens to a given port and waits for a UDP packet to arrive. The UDP Open and UDP Close VIs are only used to specify the UDP port to listen to and to free the port once the program quits. There is no connection establishment under UDP. It is the application developer’s responsibility to define the connection protocol. This can be a time consuming

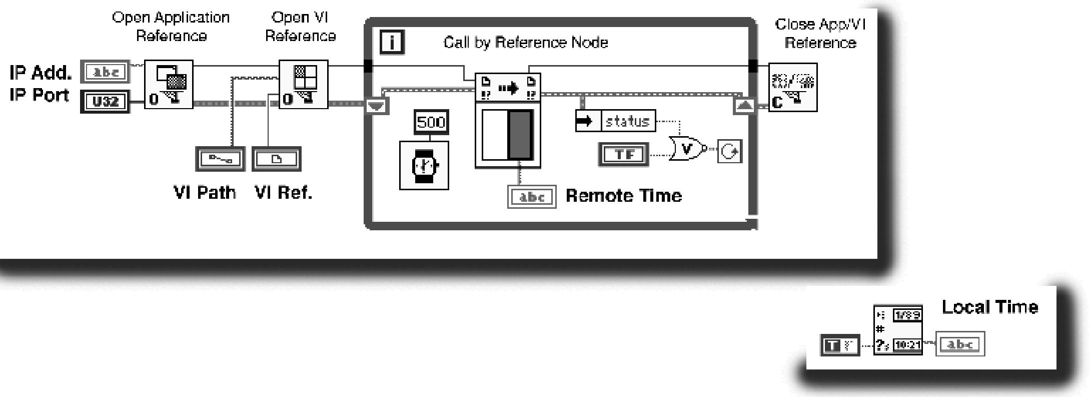


Figure 15. A simple time client-server using `CALL_BY_REFERENCE` VI. The `OPEN_APPLICATION` and `OPEN_VI` VIs open a connection with the remote LabVIEW running the specified VI (`LOCAL_TIME`). At each loop iteration, the `CALL_BY_REFERENCE` VI retrieves the remote data and displays it. Upon termination the connection is closed.

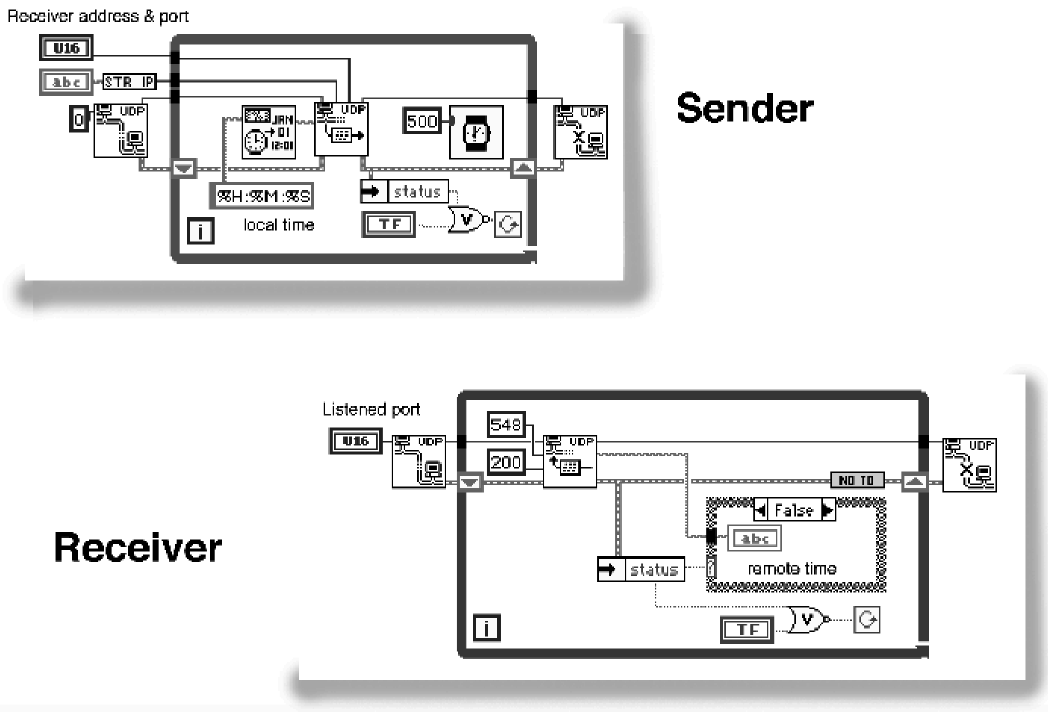


Fig. 16. A simple time client-server application using UDP. The Sender VI continuously sends its local time to the specified IP Address and IP Port. The Receiver VI listens to a given port for a given time. If a valid packet arrives, the data (remote time) are displayed. If not, it waits again for a fixed time until the user stops the VI. Upon completion the UDP ports are released.

task that requires a good knowledge of communication technology.

EXAMPLE

The electrical drive

Many mechatronic systems—i.e., those that integrate electrical and mechanical parts—used in a control engineering laboratory are attractive to students because they often yield responses that are easy to identify visually. Furthermore, experimentation can typically be conducted in a reasonable amount of time. For example, a complete laboratory experiment could take between one and two hours of work, during this period the student carries out modeling and design studies, including shorter periods (5 to 15 minutes) of interaction in real-time mode with the experiment for measurement and control purposes.

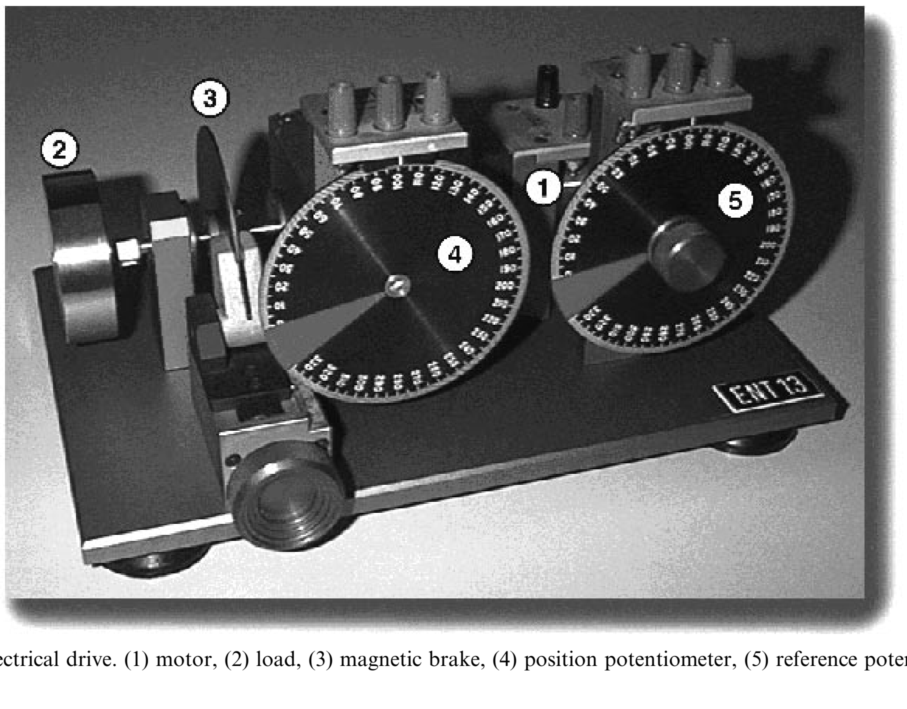


Fig. 17. Electrical drive. (1) motor, (2) load, (3) magnetic brake, (4) position potentiometer, (5) reference potentiometer.

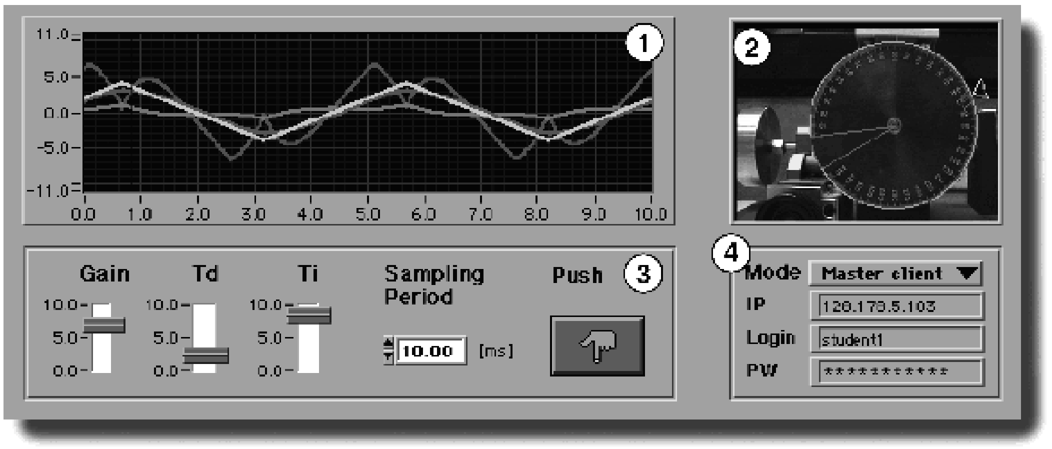


Fig. 18. Client GUI. (1) Scope area, (2) visual area with enhanced video feedback, (3) Parameters, (4) Administrative area.

The electrical drive is a typical mechatronic system (Fig. 17), which is used in many textbooks to illustrate an automatic control system. The example considered here is simple and exhibits an almost linear behaviour. It consists of a 14 W DC motor (1) equipped with a built-in tachometer. The motor drives a load—in this case a steel disk (2). An adjustable magnetic brake (3) introduces a viscous friction effect, allowing thereby a modification of the time constant during operations. Either axle angular position or speed can be controlled by adjusting the motor voltage.

The angular position is measured by a potentiometer (4) connected to the motor axle through a reduction device. The reference value can be generated manually by a similar potentiometer (5). Both potentiometers are equipped with enlarged disks, which permit easy visualisation of the motion, either locally or remotely.

Visual feedback

The visual feedback is provided by the graphical user interface (GUI). A cockpit-like metaphor [5] is used to present the different information to the user (Fig. 18). The GUI is split into four areas. The scope area (1) enables the user to follow the time evolution of all signals relevant to the experiment (for example the internal states of the controller). The visual area (2) provides the video feedback of the real process enhanced with the virtual representation of the process. The user is allowed to modify the parameters (3) of the controller as well as other adjustable characteristics of the experiment, such as the sampling period. The push button is meant for remotely perturbing the physical system. The administrative area (4) manages the different connection stages such as user login and quitting.

Augmented reality and video grabber VIs

Video feedback is especially well suited for mechatronic processes. Special attention has been made to providing the user with not only the video image of the process but also by superimposing other information such as the virtual reality representation of the physical system, resulting a composite image called **augmented reality** (Fig. 19). The virtual image is derived from the measurements made on the real system, whereas the video image comes from the video camera. Special care is needed to synchronise these two representations. This is largely compensated for by the benefits resulting from their combination.

The virtual image is also useful when the available bandwidth is small. Instead of using a large portion of the available bandwidth with the video image, only a few video images per second (1 or less) are sent. The missing dynamic of the video is compensated for by the animation of the virtual image. When using the highest compression factor, the smallest size for the video image is about 2 Kilobytes whereas for the ‘same’ information

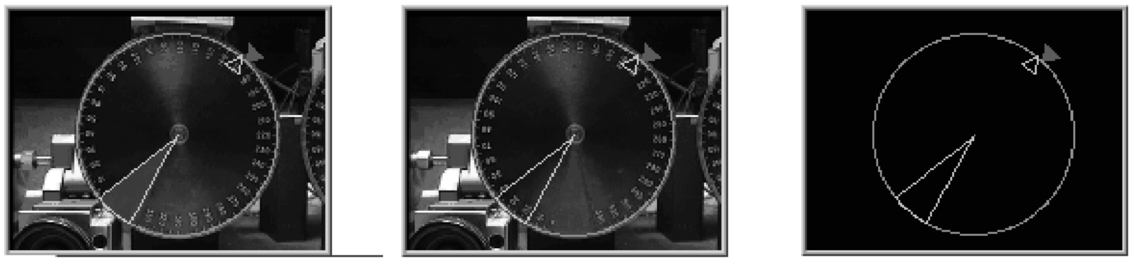


Fig 19. Video and virtual reality image. (a) Generally speaking the video image and the virtual view are synchronized. (b) The virtual view is updated more often than the video image when the transmission channel is loaded. (c) Only the virtual view is displayed when the transmission channel is heavily loaded.

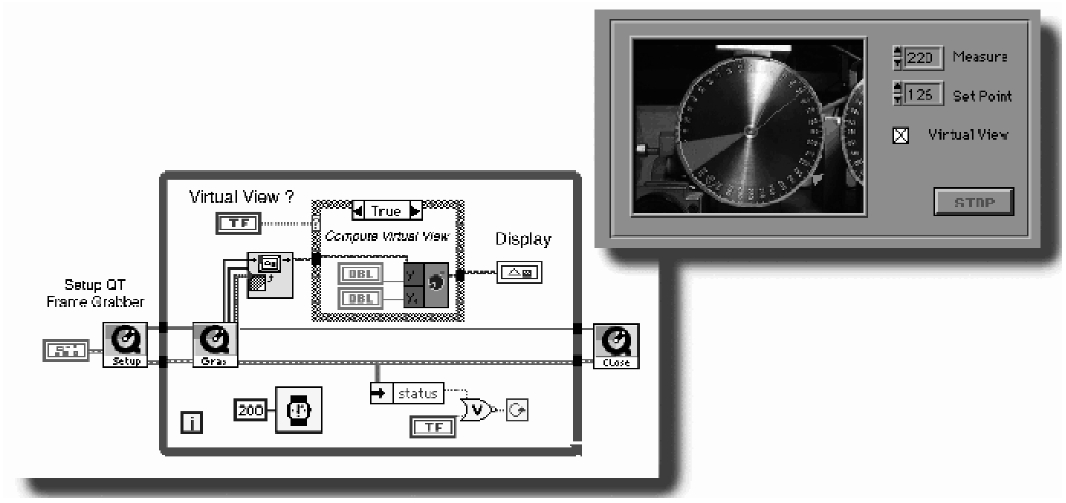


Fig. 20. Simple QT frame grabber. The video source is selected by using the `SETUP_QT` VI. The acquired image (`QT_GRAB` VI) is converted and displayed 5 times per second in a Picture indicator. The virtual view can be added to the real view of the setup. Upon completion the video source is released.

(angle), only 2 Bytes are needed to update the virtual image. This wide range of possible packet size gives considerable room for adaptation.

Observe that the video image carried far more information than the virtual image. Through the real image, the user can get a feeling for the real setup. This is essential in remote experimentation. The video image also gives environmental information which is undetectable using the virtual image, as for example a wire running across the moving parts of the electrical drive.

A set of VIs was developed to grab images coming from a video camera. These VIs are based on QuickTime (QT). They can use any input source supported by QT provided that the corresponding vdig (driver) exists. The image is displayed in a LabVIEW picture indicator. These images can be compressed in JPEG format (other formats are possible) before being sent across the Internet. On the receiver side the images are first decompressed and then displayed. If needed, the virtual image is superimposed to the video image before being displayed.

Figure 20 presents a simple video grabber with an augmented view of the real system. The image is updated 5 times per second.

CONCLUSIONS

One of the major requirements in engineering education is to provide students with convenient environments to practice what they have been taught or what they have learned conceptually. This is true in traditional education as well as in distance learning. It should be stressed that local or remote experimentation on physical systems is an essential complement to simulation and written exercises. Compared with experimentation in virtual reality, real experimentation is easier to implement and more versatile. In fact, adding or selecting another physical setup does not involve the elaboration of complex mathematical models and graphical representations. In addition, engineering students gain confidence in their ability to deal with real applications, which is going to be their career challenge.

When planning to set up an environment for experimentation in academia, the criteria for selecting a commercial solution are different from in the industry. Usually, the need for interactivity is higher and the overall capabilities are lower. The scalability and the ease in designing a user interface provided with LabVIEW, as well as its multi-platform implementation, make this package well suited to develop didactic tools. Moreover, the possibility to compile standalone virtual instruments, which can be distributed freely to the students, is a big advantage.

Various ways exist of exploiting LabVIEW for implementing automatic control solutions according with the user requirements in terms of quality of services and the constraints inherent in the controlled physical system. The most critical part is the handling of the real-time operations. The survey of different solutions given in this paper enables the selection of the solution most suitable for a particular application. Depending on the curricula into which the practice is integrated, coding the control algorithm in G, C or MATLAB may be chosen. The required cycle time is also an important element when selecting an implementation solution. Finally, when security is a major concern, embedded solutions have to be considered.

Programming is usually not the main topic of automatic control laboratory sessions. Practical training of the material learned is the main point. Thus, three different implementation approaches have been chosen at the Swiss Federal Institute of Technology. The first one is applied to engineering students spending only a few hours in the laboratory concurrently with the basic control course. They are provided with standalone VIs which enable them to experiment locally or remotely the behaviours of the controllers they have studied in class, such as the PID controller. In such a case, they use high level VIs, which let them observe the effect of changing important tuning parameters. But, they do not change the underlying control algorithms, so requiring no programming. These VIs are developed by the educators using LabVIEW enhanced with the Real-Time Kernel. The second approach concerns students involved in advanced automatic control courses who may want to validate the various control algorithms they have studied. Since they usually carried out the design and the simulation with MATLAB, they are provided with LabVIEW enhanced with the Real-Time Kernel and the MATLAB interpreter for implementation purposes. The last approach is followed by the students, which conducts a semester or a diploma project on a didactic or industrial setup. Here, as the constraints may vary, they undertook a short introductory course in real-time implementation and LabVIEW programming before choosing the most suitable solution to their application.

For distance learning purposes, the VIs are developed by a team of educators and computer scientists. From the server side, a real-time control loop is implemented with one of the available implementation solutions. To guarantee the best possible quality of service and to provide a versatile client management, the communication layer is developed using the LabVIEW UDP tools. To broadcast the view of the real experiment running remotely, a set of QT video acquisition VIs have been developed. They provide LabVIEW with a video image displayed in a picture indicator properly synchronised with the data display.

Finally, the proposed solutions demonstrate the feasibility of using LabVIEW for real-time control, allowing students to carry out experimental studies either on campus or remotely, as well as tutors to present live in-class demonstrations. Moreover, the proposed paradigm for real-time control implementation is not only limited to education. In research and industry, its ease of use also represents an interesting opportunity to meet the growing needs of scientists for fast prototyping. This paradigm enables teachers to implement real-time control solutions in a really efficient manner, both from a time and resources perspective.

REFERENCES

1. D. Gillet, G. F. Franklin, R. Longchamp and D. Bonvin, Introduction to automatic control via an integrated instruction approach, *3rd IFAC Symp. Advances in Control Education*, Tokyo, Japan, (1994) pp. 83–86.
2. G. F. Franklin, J. D. Powel and M. L. Workman, *Digital Control of Dynamic Systems*, 3rd Edition, Addison-Wesley (1997).
3. Ch. Salzmann, D. Gillet, R. Longchamp and D. Bonvin, Framework for fast real-time applications in automatic control education, *4th IFAC Symp. Advances in Control Education*, Istanbul, Turkey (1997) pp. 345–350.
4. H. A. Latchman, Ch. Salzmann, S. Thottapilly and H. Bouzekri, Hybrid asynchronous and synchronous learning networks in distance education, *Int. Conf. Engineering Education, ICEE 98*, paper 351, Rio de Janeiro, Brazil, 1998.
5. D. Gillet, Ch. Salzmann, R. Longchamp and D. Bonvin, Telepresence: an opportunity to develop practical experimentation in automatic control education, *European Control Conference, ECC 97*, paper 439, Brussels, Belgium (1997).
6. Ch. Salzmann, H. A. Latchman, D. Gillet and O. D. Crisalle, Requirements for real-time experimentation over the Internet, *Int. Conf. Engineering Education, ICEE 98*, paper 222, Rio de Janeiro, Brazil (1998).

Christophe Salzmann received his MS degree in Computer and Information Sciences in 1999 from the University of Florida, Gainesville. He is currently a Ph. D. student at the Automatic Control Institute at the Swiss Federal Institute of Technology—Lausanne (EPFL). During 1995 he was an invited scientist at National Instruments, Austin, TX, where he worked on LabVIEW. He is also responsible for the LabVIEW User Group at the EPFL and for the development of the Real-Time Kernel. His research interests include real-time computing, telepresence, distance learning, multimedia technologies and communication networks with an emphasis on bandwidth adaptation.

Denis Gillet received his Diploma in Electrical Engineering in 1988 from the Swiss Federal Institute of Technology—Lausanne (EPFL) and his Ph. D. degree in Control Systems in 1995 from the same university. During 92/93, he worked as a Research Fellow at the Information Systems Laboratory, Stanford University. In 1996, he was an invited Professor at the National Polytechnic Institute Grenoble for three months. Currently, he is an

Associate Professor (MER) at the EPFL. His research interests include on-line optimization and control, multimedia technologies, distance learning, telepresence, fast prototyping and real-time implementation.

Pierre Huguenin received his Diploma in Mechanical Engineering in 1988 from the Swiss Federal Institute of Technology—Lausanne (EPFL). He is currently a Ph. D. student at the Automatic Control Institute (EPFL). His current research interests include real-time computing, modelling, non-linear control and intelligent transportation system.