Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

WinMechLab: A Windows-based Software Tool for Real-time Control of Mechatronic Systems [⋆]

Ricardo Campa [*] **and Rafael Kelly** [**]

[*] *Instituto Tecnológico de la Laguna, Blvd. Revolución y Cuauhtémoc, Torreón, Coah., 27000, Mexico (e-mail: recampa@itlalaguna.edu.mx)*
[**] *CICESE, Carretera Tijuana–Ensenada km. 107, Ensenada, B.C., 22800, Mexico (e-mail: rkelly@cicese.mx)*

Abstract: Real-time control of industrial processes has become an important issue in the recent years. Advances in hardware and software technologies allow the use of single-processor computers to perform real–time tasks. The paper first explains the basic elements of PC–based real–time control systems. Then, it describes the general operation of WinMechLab, a real–time single–processor platform for the simulation and control of mechanisms based on MS–Windows. Finally, we show the application of this software tool in a simple control task of a direct–drive didactic robot arm.

1. INTRODUCTION

In the recent years, as the complexity of industrial automation processes has increased, the need of faster and more precise control systems has become apparent. Further, the implementation of control systems dealing with high–speed tasks requires dedicated hardware and sofware based on real–time features.

Real–time systems are those where it is absolutely imperative that their response occur within a required deadline. They arise from the intersection of control and computing engineering. In fact, control engineers need real–time systems to implement their algorithms, so most practical control systems are real–time systems.

The traditional approach to implement real–time controllers in industrial plants is to use several interconnected computers; one or more embedded processors (e.g. DSPs) are dedicated to I/O and control tasks, while a host computer (usually a PC) is used exclusively for monitoring purposes. This approach, however, is not only expensive, but requires specific hardware.

An alternative is the use of high–performance personal computers, which can implement several tasks in a single processor. These PC–based control systems are characterized by being low–cost, flexible, and able to handle plentiful software and hardware resources. Modern computing systems, with high processing rates and memory capacity, as well as the so–called *real time operating systems* (RTOS) are the key features for the effective operation of the single–processor architecture.

There exist some recent works on PC–based real–time control systems running on general–purpose RTOSs, such as QNX (Loffler *et al.* [2002], YaGuang and WenHai [2006]) and Linux (Bellini *et al.* [2002], Macchelli and Melchiorri [2002], Basso *et al.* [2005]). Microsoft Windows is not an RTOS per se, but we can use third–party real–time extensions to add it up real–time features. One of those extensions is RTX from Ardence (formerly VenturCom), which has been used under a Windows NT platform (Monroy *et al.* [2001], Campa et al. [2004], He *et al.* [2004]).

The aim of this paper is threefold. First, we illustrate distinctive concepts of real–time control systems under the MS–Windows operating system. Second, we present WinMechLab 2.0, a general purpose computing system for real-time control of mechanisms that we have developed in the Robotics Laboratory at CICESE Research Center (Campa et al. [2004]). Release 2.0 includes some new features, such as the ability of running under latest Windows versions, and sharing information among computers (see Monroy et al. [2007]). Third, we show the results of testing the performance of this system in the control of a direct–drive didactic robot arm.

2. WINDOWS–BASED REAL–TIME CONTROL

2.1 RTOS generalities

Important features of real–time systems are concurrent activities, interprocess communication, and especially timing requirements which lead to deterministic and predictable behavior. To fulfill these demands in a personal computer, it is necessary to employ a real–time operating system.

In general, a RTOS requires:

- Multitasking:
  The ability to execute several tasks in an apparently simultaneous fashion.
- Process' threads that can be prioritized:
  Each concurrent task is executed as a separate block of information, which is called a thread. RTOSs let to assign a priority to each thread, so as to control the execution order.

[⋆] This work was partially supported by CONACyT (grants 45826 and 60230), Mexico.

- A sufficient number of interrupt levels:
  RTOSs use a program called the *scheduler* to interrupt the execution of the current thread and figure out which other to give control to next.

In general–purpose operating systems, task scheduling depends of several factors and is commonly imprecise. In a RTOS, however, scheduling must be done according to a very rigid, preestablished criterion, so as to fulfill the assigned timing for each task.

Some RTOSs are created for special (embedded) applications; others are for more general purpose. To the latter kind belong QNX and RT-Linux. A drawback of these RTOSs, however, is that they are utilized typically by specific research groups but not by the common user.

It is worth mentioning here the Matlab/Simulink/RTW platform from Mathworks. This is a a well–known computer aided control system design (CACSD) tool that provides a generic interface to the Real–Time Windows Target (RTWT), a small real–time kernel using the built–in PC clock as its primary source of timing (Mathworks [1999]). However, as pointed out by Gambier [2004], the RTWT lacks from multitasking/multithreading support, and is not considered an RTOS.

2.2 A real–time extension for Windows

Since the appearing of Windows 1.0 by mid 1980's, Microsoft has taken two parallel routes in operating systems. On the one hand, there are "home versions" focusing on simple applications for the common user; on the other, the "professional versions", directed to the IT user, have more specialized features, such as networking and security.

Windows NT 1.0 (for New Technology) was the first professional version of Windows; it is based on a new kernel and introduces preemptive multitasking. This technology has been preserved in the newer versions of Windows (Windows 2000, Windows XP and Windows Vista are, in fact, versions 5.0, 5.1 and 6.0 of Windows NT). For this reason, in the subsequent, the term *Windows NT* will be used to refer to any of these versions.

All of Windows NT operating systems have a model based on processes and threads. Another important feature of NT is its layered structure, from the low–level Hardware Abstraction Layer (HAL), up to the several high–level NT subsystems, designed to implement abstract services such as emulating different target operating systems.

Due to these features, the Windows NT family of operating systems becomes a platform which is appropriate for the development of systems where it is required a major control of hardware resources together with the flexibility of a graphical environment. Windows NT functionality, however, is not targeted for "hard" real–time applications, and thus, the use of Windows NT is significantly restricted, and often prevented, for these applications (VenturCom [1999]).

To overcome this limitation, some software companies have developed commercial modules that can be installed over Windows NT to add it up some hard real–time features. One of those modules is RTX (standing for Real–Time eXtension) from Ardence (formerly VenturCom), which we have chosen as the platform for implementing real–time tasks under Windows NT.

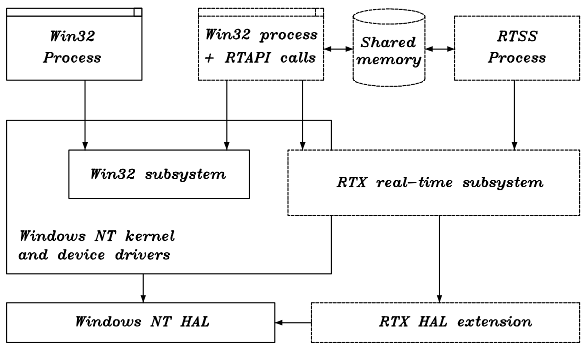


Fig. 1. Windows NT plus RTX architecture.

RTX enables Windows NT to function as both a general–purpose operating system and a high–performance RTOS on the same computer. This is possible thanks to RTX's feature of handling two different kinds of real–time processes inside Windows NT (VenturCom [1999]):

- Win32 Processes.
  They use a subset of the standard Windows API (Win32) to keep the advantage of using GUIs and other Win32–only functions, but they also can use some RTX real–time functions.
- RTSS Processes.
  They are executed under the Real-Time Subsystem (RTSS) which is installed by RTX. They use their own scheduler, so that RTSS processes can achieve hard real–time tasks.

Figure 1 shows a general scheme of the Windows NT architecture, including RTX. Other key real–time features of RTX are:

- High execution rates for periodic RTSS processes (10 kHz).
- Complete range of RTSS process priorities, from the lowest 0 to the highest 127.
- Interprocess communication (among RTSS and/or Win32 processes) via shared memory.

3. WINMECHLAB: A MECHATRONICS LABORATORY FOR WINDOWS

3.1 General description

WinMechLab is an acronym standing for *Windows Mechatronics Laboratory* and it is a general–purpose system for the real–time control of mechanisms, running under MS–Windows, plus RTX. WinMechLab permits the edition, compilation, simulation, and execution of control algorithms, written using a simple syntax, very similar to the standard C language.

One of the main features of WinMechLab is its versatility in handling hardware devices. WinMechLab 2.0 includes interface libraries for some commonly used acquisition boards: MultiQ and MultiQ-PCI from Quanser Consulting, ServoToGo from ServoToGo Inc., and MFIO-3A from Precision MicroDynamics. New acquisition boards can be added, provided that I/O C–Language API functions for those boards exist.

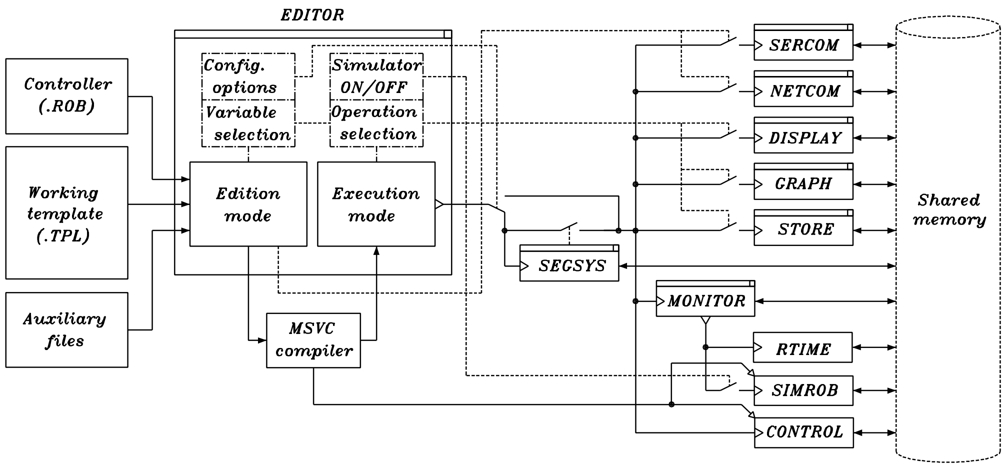


Fig. 2. General block diagram for the operation of WinMechLab 2.0.

Besides, the kind of mechanisms that can be controlled by WinMechLab is also diverse. Currently, this software tool is used with several direct–drive mechanical arms in the Robotics Laboratory at CICESE Research Center.

Another interesting feature of WinMechLab is its ability for simulation of mechanisms, only providing a proper file containing the corresponding model in state–variable description. Simulations obtained using WinMechLab match pretty well to experimental results of our robotic systems.

Some additional features which are new in version 2.0 are the ability of executing external real–time processes, and the possibility of communicating two or more computers via serial port or a TCP/IP connection. An interesting application of these features can be found in (Monroy et al. [2007]).

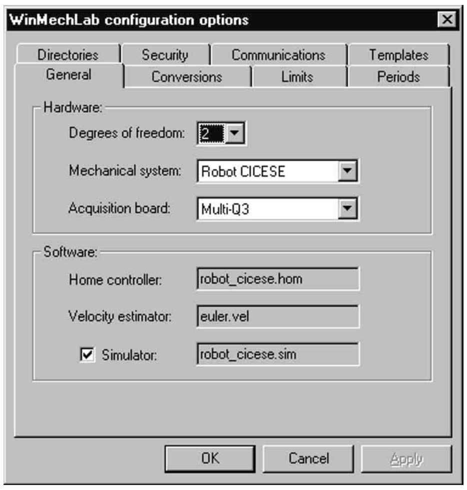
3.2 WinMechLab operation

Figure 2 shows a general diagram of WinMechLab's operation. The main program (EDITOR) is where the user can edit, compile and execute control algorithms. EDITOR includes a configuration panel (shown in Figure 3) where the user can choose the mechanism to control, the acquisition board, sampling periods, and communication parameters, among many other things.

WinMechLab's Editor operates in two modes:

- Edition Mode.
  In this mode the user can edit the control algorithm (.ROB file) by using the whole functionality of Windows environment. In addition, the user can select the variables he wants to manipulate during the execution of the algorithm, and configure the general behavior of WinMechLab.
- Execution Mode.
  In this mode the user can start the execution of the real–time processes needed to achieve the control task. Before execution, the user can choose the operations (numerical or graphical display, file storage, serial or TCP/IP communications) to carry out during execution stage.

The .ROB file is written using a C-Language syntax, but with a very simple format. As an example, Figure 4 shows a program for implementing a PID controller in WinMechLab. Note that only the equations of the control algorithm are required (they are executed iteratively every Sampling_period seconds). Joint positions, velocities and torques are handled internally by WinMechLab.


WinMechLab configuration options

Hardware:
Degrees of freedom: 2
Mechanical system: Robot CICESE
Acquisition board: Multi-Q3

Software:
Home controller: robot_cicese.hom
Velocity estimator: euler.vel
☑ Simulator: robot_cicese.sim

Fig. 3. WinMechLab configuration window.

```
/* ******** PID CONTROLLER FOR 2-DOF PLANAR ROBOT ********* */

/* **************** Variable declarations *************** */

float pos1, pos2;              // Joint positions [deg]
float vel1, vel2;              // Joint velocities [deg/s]
float tau1, tau2;              // Applied joint torques [Nm]

float kp1=2.1, kp2=1.4;        // Proportional gains [N.m/deg]
float kd1=0.6, kd2=0.3;        // Derivative gains [N.m.s/deg]
float ki1=2.4, ki2=0.9;        // Integral gains [N.m/(s.deg)]

float posd1=45.0, posd2=10.0;  // Desired positions [deg]
float we1, we2;                // Integrals of position errors
float we1a=0, we2a=0;          // Previous values of wei

/* ****************** Control algorithm ***************** */

// Integrals of position error (Euler algorithm)
we1 = we1a + Sampling_period*(posd1-pos1);
we2 = we2a + Sampling_period*(posd2-pos2);

// PID control law (regulation case)
tau1 = kp1*(posd1-pos1) - kd1*vel1 + ki1*we1;
tau2 = kp2*(posd2-pos2) - kd2*vel2 + ki2*we2;

// Updating integral variables (for next iteration)
we1a = we1;
we2a = we2;
```

Fig. 4. WinMechLab program for PID control

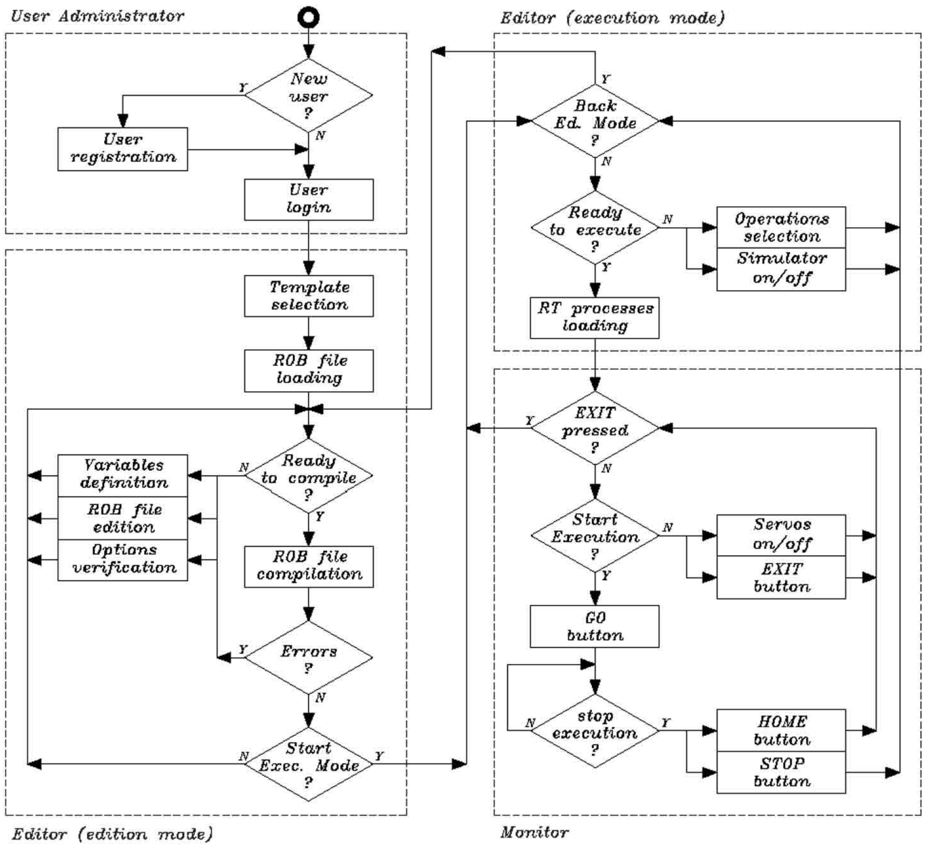


Fig. 5. User's flowchart for WinMechLab 2.0.

To enter Execution Mode, the .ROB file must be first compiled without errors. After the compilation, a real–time program containing the control algorithm (CONTROL) is generated. This program will be executed as an RTSS process during the execution of the task.

Once in Execution Mode, the user can select among five types of operations to carry out with the selected variables, during the execution of the control algorithm: to display them in numerical form, in graphical form, to store them in a text file, or to send them to another computer via a serial or TCP/IP port. These operations make use of real–time processes of Win32 type, with relatively low priorities, and execution periods selected by the user. Those optional Win32 processes are:

- DISPLAY: Allows the numerical visualization of up to twelve variables and six parameters.
- GRAPH: For the graphical display of variables during execution. There can be up to six instances of this process, running simultaneously.
- STORE: For the storage of up to twelve variables to a file, in ASCII format.
- SERCOM: For communication to another computer, via serial port (RS232).
- NETCOM: For communication via Ethernet, using TCP/IP.

Besides these processes for variable manipulation operations, there are other high priority RTSS processes that are executed simultaneously:

- RTIME: Its unique function is to continuously keep the count of the real–time counter. This process is executed every 0.1 ms, and given its importance, it has the highest priority (127).
- CONTROL: This is the real–time process that was generated after compiling the .ROB file. It is the second in order of priority (126) and its execution period is assigned by the user (default is 2.5 ms).
- SIMROB: It is an optional process that corresponds to the simulator of the selected mechanism. It is executed every 0.2 ms, with a priority of 125.

There exist another real–time process, called MONITOR, which has some important functions, such as to display the real time counter, to turn on and off the servomotors, and to start and stop the execution of the controller.

Figure 5 shows the flowchart indicating the steps that a registered user should follow to load, configure and execute a real–time controller using WinMechLab 2.0. Notice that the user can, at any time, go back to the edition of the control algorithm, after running the controller. Moreover, the user can easily commute between the execution of a simulation or a real experiment. These features are very useful during the controller design and gain tuning stages.

As an additional feature of WinMechLab, it includes a graphic viewer to easily display, edit, and convert to EPS format, the data resulting from a simulation/experiment.

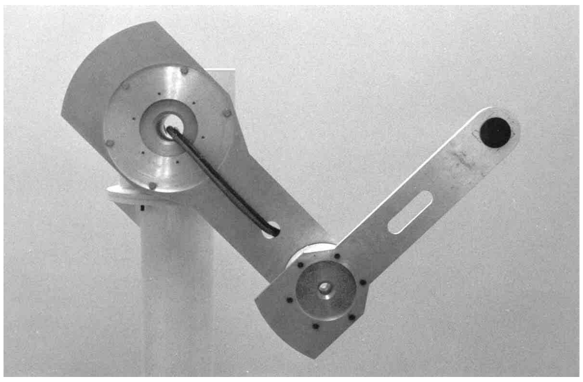


Fig. 6. Experimental arm

4. AN APPLICATION TO ROBOTICS

In order to illustrate the operation of WinMechLab, we carried out some experiments of joint motion control on a two–DOF direct–drive didactic arm that is in the Robotics Laboratory of CICESE Research Center (see Figure 6).

4.1 Mechanical system

This robot arm is made of two rigid links and two high–torque, brushless direct–drive servos (models DM1200A and DM1015B from Parker Compumotor), which are operated in *torque mode*, so the motors act as a torque source and they accept an analog voltage as a reference of torque signal. Position information is obtained from incremental encoders located on the motors, and velocity information is obtained by numerical differentiation of the position signals.

The dynamics of our robot arm has the general structure (Kelly et al. [2005]):

$$M(\boldsymbol{q})\ddot{\boldsymbol{q}} + C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau} \tag{1}$$

where, in our application, $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$ are the 2×1 vectors of joint displacements, velocities and accelerations, respectively; $\boldsymbol{\tau}$ is the 2×1 vector of applied torque inputs, $M(\boldsymbol{q})$ is the 2×2 symmetric positive definite manipulator inertia matrix, $C(\boldsymbol{q},\dot{\boldsymbol{q}})$ is the 2×2 matrix of centripetal and Coriolis torques, and $\boldsymbol{g}(\boldsymbol{q})$ is the 2×1 vector of gravitational torques due to gravity.

The physical parameters and complete dynamic model of this arm are reported by Reyes and Kelly [1997].

4.2 Motion control

Let $\boldsymbol{q}_d$, $\dot{\boldsymbol{q}}_d$, $\ddot{\boldsymbol{q}}_d$, be the desired joint position, velocity and acceleration vectors of a motion trajectory, which are chosen as bounded functions. The primary goal of motion control in joint space is to make the robot joint positions $\boldsymbol{q}$ track a given time–varying $\boldsymbol{q}_d$. Rigorously, the motion control objective in joint space is achieved provided that

$$\lim_{t\to\infty} \widetilde{\boldsymbol{q}}(t) = \boldsymbol{0} \tag{2}$$

where $\widetilde{\boldsymbol{q}}(t) = \boldsymbol{q}_d(t) - \boldsymbol{q}(t)$ is the joint position error. Similarly, $\dot{\widetilde{\boldsymbol{q}}}(t) = \dot{\boldsymbol{q}}_d(t) - \dot{\boldsymbol{q}}(t)$ is the joint velocity error.

For the purposes of this paper, we consider the *computed–torque control* which is one of the simplest motion control schemes, whose practical effectiveness has been reported for more than two decades. The computed–torque controller is given by (Kelly et al. [2005]):

$$\boldsymbol{\tau} = M(\boldsymbol{q})[\ddot{\boldsymbol{q}}_d + K_v\dot{\widetilde{\boldsymbol{q}}} + K_p\widetilde{\boldsymbol{q}}] + C(\boldsymbol{q},\dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}). \tag{3}$$

where, besides the previously defined terms, K_p and K_v are 2×2 symmetric positive definite gain matrices.

If the manipulator dynamic model is exact, then the computed–torque controller (3), applied to the robot dynamics (1), achieves the decoupling of all the joints, resulting in an asymptotically stable linear time–invariant closed–loop system, in terms of the error $\widetilde{\boldsymbol{q}}$, and thus asymptotically exact tracking is ensured (see Kelly et al. [2005]).

4.3 Experimental results

An important issue for experimental evaluation of robot control algorithms is the choice of the desired trajectories $\boldsymbol{q}_d(t)$ for the robot motion. The expression of the desired trajectory used in this project is similar to those proposed in Dawson *et al.* [1994], that is

$$\boldsymbol{q}_d(t) = \begin{bmatrix} a_1[1 - e^{-\alpha_1 t^3}] + b_1[1 - e^{-\alpha_1 t^3}]\sin(\omega_1 t) \\ a_2[1 - e^{-\alpha_2 t^3}] + b_2[1 - e^{-\alpha_2 t^3}]\sin(\omega_2 t) \end{bmatrix} \tag{7}$$

where b_1 and b_2 denote the amplitude of the steady state sine functions whereas ω_1 and ω_2 represent the angular frequencies of the desired trajectory for the shoulder and elbow joints respectively. For the experimental session we used the next values: $a_1 = 45°$, $a_2 = 60°$, $b_1 = 10°$, $b_2 = 125°$, $\alpha_1 = 2$ s^{-1}, $\alpha_2 = 2$ s^{-1}, $\omega_1 = 15$ rad/s, $\omega_2 = 3.5$ rad/s.

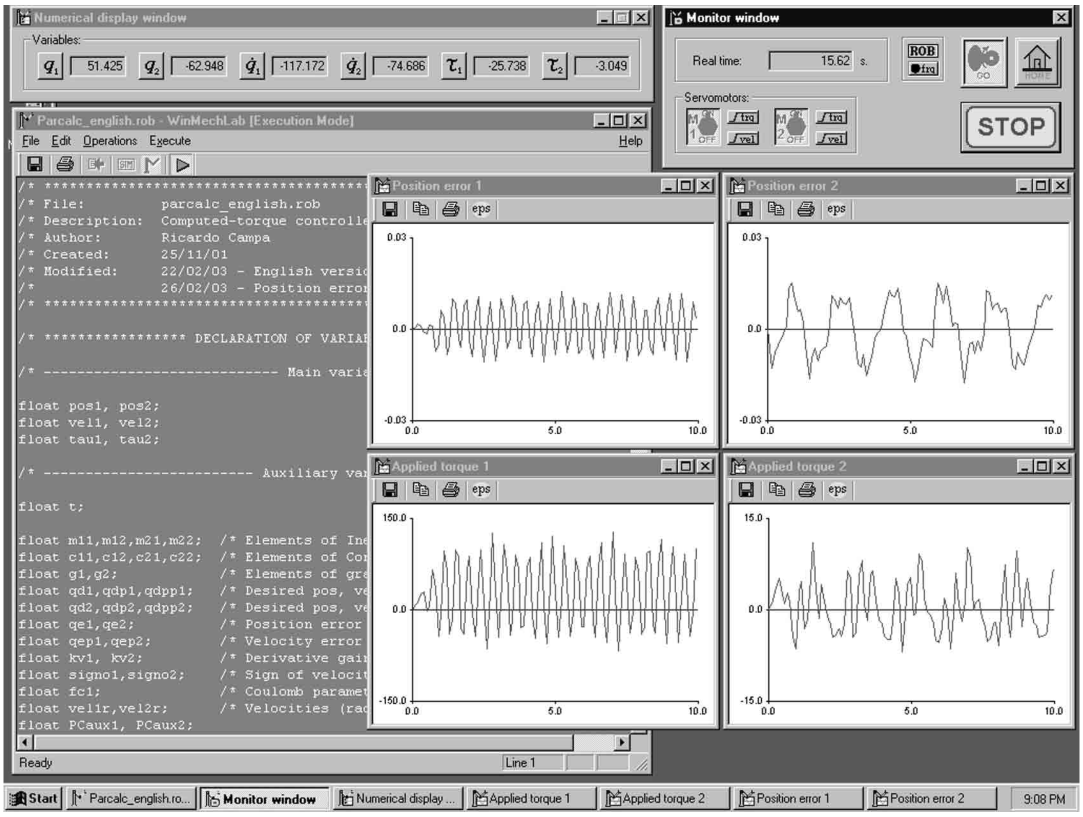
The controller (3) was implemented together with trajectory (7) as a real–time process, and executed at a sampling period of 2.5 [ms] using WinMechLab. The values of the controller gains used for the experiments were: $K_p = \text{diag}\{400, 1000\}$ [Nm/rad], and $K_v = \text{diag}\{40.0, 63.2\}$ [Nms/rad].

Figure 7 presents the computer screen during the experimental testing. Notice the windows for EDITOR (showing the control algorithm), MONITOR, DISPLAY and GRAPH processes. In the same figure we can see the graphs of the joint position errors (in radians) and applied torques (in Newton-meters). Although the errors are not zero as ideally (see equation (2)), they are small enough, indicating a good performance of the controller. Non–null errors are considered to be due to the presence of unmodeled dynamics and noise in the velocity estimation (using simple Euler approximation).

5. CONCLUDING REMARKS

Thanks to the progress during the last decade in high–performance hardware and real–time software, it is possible nowadays to develop real–time control systems in single–processor computers.

Real–time operating systems play an important role in the development of PC–based controllers. MS–Windows has the advantage of being very familiar for most of the users, even though it is not a hard real–time operating system.



q_1 51.425 q_2 -62.948 $\dot{q}_1$ -117.172 $\dot{q}_2$ -74.686 τ_1 -25.738 τ_2 -3.049

Real time: 15.62 s.

```
/* ***************************************
/* File:         parcalc_english.rob
/* Description:  Computed-torque controlle
/* Author:       Ricardo Campa
/* Created:      25/11/01
/* Modified:     22/02/03 - English versio
/*               26/02/03 - Position error
/* ***************************************

/* **************** DECLARATION OF VARIAB

/* --------------------------- Main varia

float pos1, pos2;
float vel1, vel2;
float tau1, tau2;

/* ------------------------ Auxiliary var

float t;

float m11,m12,m21,m22;  /* Elements of Ine
float c11,c12,c21,c22;  /* Elements of Cor
float g1,g2;            /* Elements of gra
float qd1,qdp1,qdpp1;   /* Desired pos, ve
float qd2,qdp2,qdpp2;   /* Desired pos, ve
float qe1,qe2;          /* Position error
float qep1,qep2;        /* Velocity error
float kv1, kv2;         /* Derivative gair
float signo1,signo2;    /* Sign of velocit
float fc1;              /* Coulomb paramet
float vel1r,vel2r;      /* Velocities (rad
float PCaux1, PCaux2;
```

Fig. 7. Example of experimental session using WinMechLab

To solve this limitation and still take profit of the graphic environment of Windows, we have used Ardence's real–time extension (RTX).

In order to test the benefits of this approach we have developed a software system for real–time control of mechanisms, which is called WinMechLab. This system allows the execution of concurrent tasks, and its performance fits the requirements of real–time control in fast processes such as robot manipulators. This allows WinMechLab to be an excellent tool for learning real–time control features at a practical level.

REFERENCES

M. Basso, R. Bucher, M. Romagnoli and M. Vassalli. Real–time control with Linux: A web services approach. *Proc. of the 44th IEEE Conference on Decision and Control*, Seville, Spain, December 2005.

C. Bellini, F. Panepinto, S. Panzieri, and G. Ulivi. Exploiting a real–time Linux platform in controlling robotic manipulators. *Proc. 15th IFAC World Congress*, Barcelona, Spain, 2002.

R. Campa, R. Kelly and V. Santibáñez. Windows-based Real-time Control of Direct-drive Mechanisms: Platform Description and Experiments, *Mechatronics*, Vol. 14, pp. 1021-1036, 2004.

D.M. Dawson, J.J. Carroll, and M. Schneider. Integrator backstepping control of a brush dc motor turning a robotic load. *IEEE Transactions on Control System Technology*, Vol. 2, No. 3, pp. 233-244, 1994.

A. Gambier. Real–time control systems: A tutorial. *Proc. of the 5th Asian Control Conference*, Melbourne, Australia, July 2004.

J. He, R. Yang, Q. Zhao, and C. Wang. A chess–playing robot control system based on Windows NT+RTX. Robotica, Vol 22, pp. 339-343.

R. Kelly, V. Santibáñez and A. Loría. *Control of Robot Manipulators in Joint Space.* Springer-Verlag, 2005.

M. Loffler, N. Costescu, and D. Dawson. Qmotor 3.0 and the Qmotor Robotic Toolkit: A PC-based control platform. *IEEE Control Systems*, Vol. 22, No. 3, pp. 12-25.

A. Macchelli, and C. Melchiorri. A real–time control system for industrial robots and control applications based on real–time Linux. *Proc. 15th IFAC World Congress*, Barcelona, Spain, 2002.

The Mathworks Inc. *Real–time Windows Target: For use with Real–time Workshop. User's Guide.* The Mathworks, Inc., 1999.

C. Monroy, R. Campa, and R. Kelly. An application of real–time control systems to robotics. *Robotica*, Vol. 19, pp. 323-329, 2001.

C. Monroy, R. Kelly, M. Arteaga, and E. Bugarin. Remote visual servoing of a robot manipulator via Internet2. *Journal of Intelligent and Robotic Systems*, Vol. 49, No. 2, pp. 171-187, 2007.

F. Reyes, and R. Kelly. Experimental evaluation of identification schemes on a direct drive robot. *Robotica*, Vol. 15, pp. 563–571, 1997.

VenturCom Inc. *RTX 4.3 User's Guide.* VenturCom, Inc., 1999.

K. YaGuang, and W. WenHai. Design of real–time control software based on QNX. *Proc. of the 32nd IEEE Annual Conference on Industrial Electronics*, Paris, France, November 2006.