Robotica (2001) volume 19, pp. 323–329. Printed in the United Kingdom © 2001 Cambridge University Press

An application of real-time control systems to robotics*

Carmen Monroy, Ricardo Campa and Rafael Kelly

División de Física Aplicada, CICESE, Apdo. Postal 2615, Adm. 1, Ensenada, B.C., 22800 (MEXICO).
E-mail: rkelly@cicese.mx

(Received in Final Form: September 26, 2000)

SUMMARY
This paper illustrates basic concepts of real-time control systems through the application of a real-time single-processor computing environment for the control of a robotic arm. The paper describes elements for the selection of the real-time architecture, the control algorithm and the graphical user interface. The system provides an opportunity for users to verify the robot performance by changing on-line the controller parameters and the shape of the desired motion.

KEYWORDS: Robot control; Real-time control systems; Graphical user interface; Computed torque control.

1. INTRODUCTION

Besides robotics is nowadays of high importance as a technological tool in a number of automation processes, it also appears as an attractive discipline to illustrate important engineering concepts involved in control systems and computer sciences.

A well suited definition capturing the essence of robots is that evoked by Brady[1] which establishes that "a robot is a surprisingly animate machine". To this family belong the mobile robots – basically legged and wheeled machines – and robot manipulators. The latter class, being by far the most utilized in industrial applications, is of our concern in this paper.

Robot manipulators offer interesting theoretical and practical challenges to control system designers. The nonlinear nature of their dynamical behavior is well suited for the application of advanced control techniques such as nonlinear control. On the other hand, implementation of control systems for robot manipulators dealing with high speed tasks, requires dedicated hardware or control software based on real-time features.

Real-time systems arise from the intersection of control and computing engineering. A real-time system is one for which it is absolutely imperative that the response occurs within the required deadline. Control engineers need real-time systems to implement their systems, so most practical control systems are real-time systems.[2]

The objective of this paper is to illustrate distinctive concepts of real-time control systems using as a testbed a robotic arm. To this end, a real-time control system implemented on a single-processor computer has been developed based on Windows NT's real-time extension,[3] RTX, together with a graphical user interface built using the TILCON Real-Time Developer.[4] This system widely exploits the features of real-time systems such as concurrent processes executed at prescribed rates and interprocess communication.

The controlled real-world system is a direct-drive mechanical arm formed by two joints (see Figure 1). A direct-drive arm is a mechanical arm in which the shafts of actuated joints are directly coupled to the rotors of motors with high torque.[5] The "computed torque control" – which is a well known model-based control scheme for motion control of manipulators in joint space[6] – has been utilized as control algorithm. It was coded in *C* language and executed every 2.5 [msec] on a Windows NT platform thanks to the deterministic responses of the RTX module.

2. REAL-TIME CONTROL ARCHITECTURE

Important features of real-time systems are concurrent activities, interprocess communication, and especially, timing requirements which lead to a deterministic and predictable behaviour. Real-time systems must respond accurately to internal and external events. Events can be periodic or nonperiodic but each event requires a certain amount of processing time and has a defined deadline.

In practice, control algorithms are coded as digital control programs which must be executed at specified rates. Depending on the nature of the controlled real process, the rates are ranged between several Hertz and thousand of

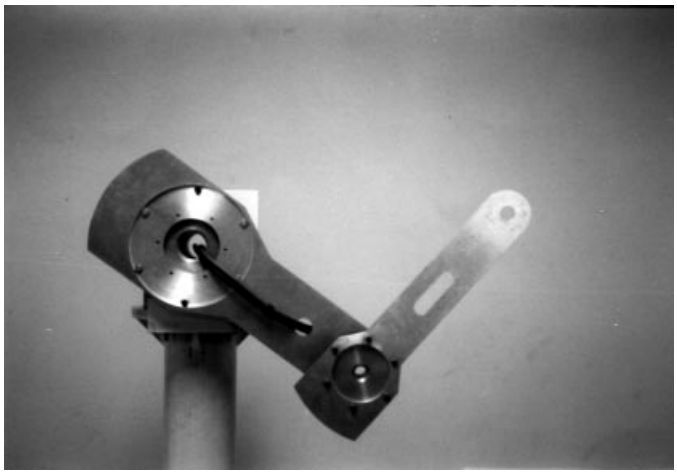

Fig. 1. Experimental arm.

*Work partially supported by CONACyT grant 225080-5 32613-A and project CYTED.

https://doi.org/10.1017/S0263574700003167 Published online by Cambridge University Press

Hertz. The critical situations – from implementation viewpoint – occur for those demanding high sampling rates.

There are typically two hardware platform architectures to implement real-time control systems:[7]

- The traditional approach consists of two single-processor computers as illustrated in Figure 2: a general purpose computer called the "host" and an embedded one – interfaced to the host via the host's bus or a serial communication link – based, for example, on a Digital Signal Processor (DSP) equipped with an acquisition board. The control program is developed in the host and then downloaded to the DSP computer using a cross-development system. The control algorithm is executed in the DSP computer at the specified rate and data is quickly – but typically at a slower rate – transferred to the host where a graphical user interface is executed allowing the user to perform monitoring functions, start and stop the control algorithm, and modify the control gains.
- The second approach is based on one single-processor computer equipped with an acquisition board (without any processor included) as depicted in Figure 3. The computer system is based on either a real-time environment or a Real-Time Operating System (RTOS) in order to keep the critical deterministic timing requirements of the control algorithms.

The control program as well as the graphical user interface are both executed in the single-processor system. Thanks to the real-time environment or the RTOS, the control algorithm can be executed in the computer at the specified high rate, and in a concurrent way – but at low priority and rate – the data are transferred via the interprocess communication procedures to the graphical user interface where monitoring and control gain modification can be achieved on-line.

Nowadays, the availability of real-time environments for MS-Windows such as RTX, Hyperkernel, WinRT, and, on the other hand, PC-compatible RTOS such as QNX, LynxOS, RTLinux, allow the use of PC-compatible single CPU cards to have deterministic responses. Hence, control programs demanding to run at high frequencies with deterministic response, and non-real-time processes – graphical user interfaces – can be executed concurrently on one processor.

Combining the greater processing power of actual PC-compatible CPUs with either real-time environments or RTOSs, provides the basis for a single-processor architecture that can hold all the functionality of multiprocessor architectures. This is the main reason why we have decided to use the approach based on a single computer architecture supported by a real-time environment for the implementation of our real-time control system.

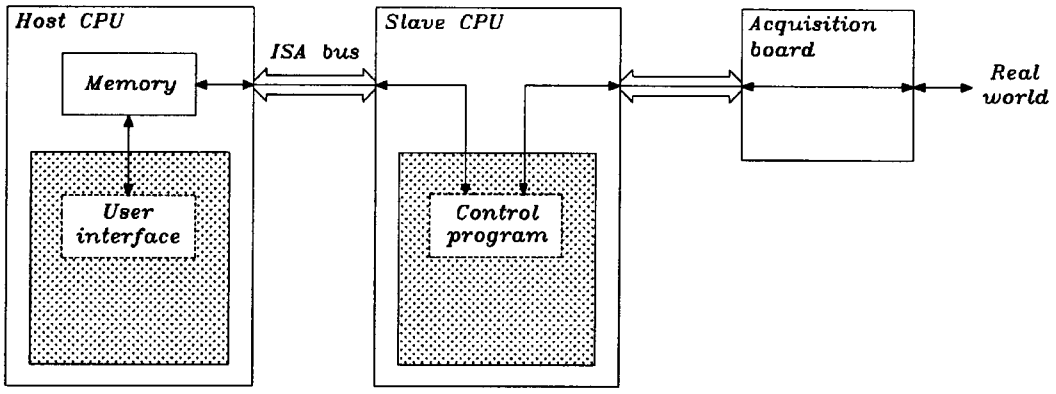

Fig. 2. Traditional two-loops real-time control system.

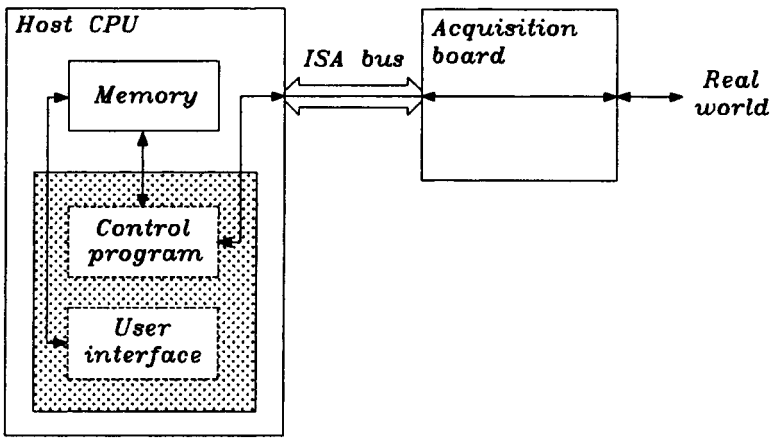

Fig. 3. Single-CPU real-time control system.

https://doi.org/10.1017/S0263574700003167 Published online by Cambridge University Press

Having reviewed the existing real-time software technology, it has been decided to use MS Windows NT operating system together with VenturCom's real-time extension, RTX, as the software platform. RTX enables Windows NT to function as both a general-purpose operating system and a high-performance real-time operating system on the same computer. Also, RTX offers key real-time features such as

- High execution rates for periodic processes (10 kHz)
- Complete range of process priorities from the lower level 0 to the higher 127
- Interprocess communication by shared memory

The hardware is completed by a ServoToGo's I/O card (without any processor included) installed into the host's ISA bus. The I/O facilities of the card that we are using are encoder inputs, analog outputs, and digital outputs.

Our application to robotics consists of controlling the motion of a mechanical arm (see Figure 1) utilizing a certain control algorithm. The desired motion of the arm as well as the control algorithm possess parameters related to the desired motion shape and the accuracy of the arm tracking the desired motion. Thus, these parameters have important meaning for the user who may want to modify them. Also, monitoring variables of the arm such as applied torques, joint velocities, and position tracking errors are of crucial importance.

Four concurrent processes have been developed for our application (see Figure 4):

- The first process, **R_TIME**, is a real-time periodic application written in *C++* language devoted to generate the clock signal. This process must be executed every 2.5 [msec] at the highest priority 127. The real-time variable **real-time** containing the elapsed time is then updated every 2.5 [msec] and used by both the control algorithm and the graphical user interface.
- The second process, **CONTROL**, is also a real-time periodic application written in *C++* language dealing with the control algorithm. Basically, this application performs the following sequence of actions:
  - Joint positions are "read" through the encoder inputs of the I/O card.
  - Joint velocities are estimated and then the control actions are computed according to the control algorithm which will be described later on.

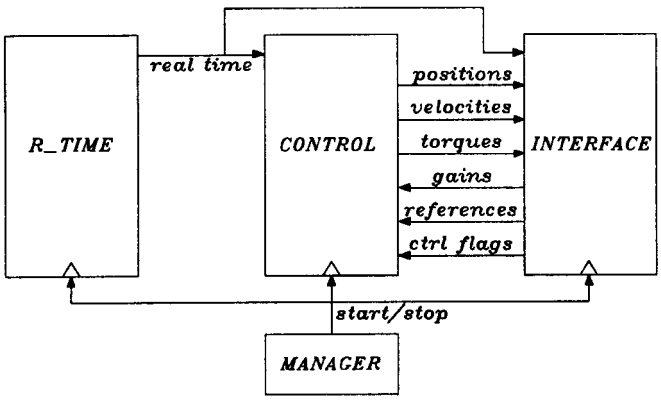

Fig. 4. Block diagram of the whole software system.

Control actions are "written" in the I/O card through the analog outputs.

These events are synchronized by the **real-time** variable, hence the application **CONTROL** runs every 2.5 [msec] at high priority 126. A number of variables such as joint positions, joint velocities and applied torques are shared with the graphical user interface. Also variables corresponding to the controller gains and the shape parameters of the desired motion come from the interface to be used in the control algorithm.

- The third process, **INTERFACE**, is a non-real-time non-periodic application written in *C++* language serving as a communication bridge with the Graphical User Interface (GUI). The GUI has been generated using the TILCON's Real-Time Graphics Editor which is a tool for designing object-oriented and event driven GUIs.[4] This application will be described later in this paper.
- The fourth process, **MANAGER**, is a low priority application written in *C++* language invoked every 0.5 [sec]. The role of **MANAGER** is to start and stop the previous described processes by using the **start/stop** variable shared with all the processes.

3. MECHANICAL SYSTEM

A vertical direct-drive arm with two rigid links (see Figure 1) has been designed and built to enable real-time control experiments. The arm links were made of 6061 aluminium, and in extent they are 0.98 [m] long from shoulder axis to the tip.

High-torque, brushless direct-drive servos are used to drive the joints without gear reduction. The motors used in the experimental arm are the models DM1200-A and DM1015-B from Parker Compumotor for the shoulder and elbow joints respectively. For this application the servos are operated in "torque mode", so the motors act as torque sources and they accept an analog voltage as a reference of torque signal. According to the actuator manufacturer, in this configuration the DM1200-A motor is capable of delivering a maximum torque of 200 [Nm], and the DM1015-B motor delivers up to 15 [Nm].

Position information is obtained from incremental encoders located on the motors, which have a resolution of 1024000 [pulses/rev] for the first motor and 655360 [pulses/rev] for the second one, and velocity information is obtained by numerical differentiation of the position signals.

The dynamics of our robot arm has the general structure:[6]

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

where in our application $\boldsymbol{q}$ is the 2×1 vector of joint displacements, $\dot{\boldsymbol{q}}$ is the 2×1 vector of joint velocities, $\boldsymbol{\tau}$ 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, $\boldsymbol{g}(\boldsymbol{q})$ is the 2×1 vector of gravitational torques due to gravity, and $\boldsymbol{f}(\dot{\boldsymbol{q}})$ is the 2×1 vector of friction torques.

Considering the values of the physical parameters of our robot arm, the corresponding entries of the robot dynamics are:[8]

https://doi.org/10.1017/S0263574700003167 Published online by Cambridge University Press

$$M(\boldsymbol{q}) = \begin{bmatrix} 2.351 + 0.168\cos(q_2) & 0.102 + 0.084\cos(q_2) \\ 0.102 + 0.084\cos(q_2) & 0.102 \end{bmatrix}, \quad (2)$$

$$C(\boldsymbol{q}, \dot{\boldsymbol{q}}) = \begin{bmatrix} -0.168\sin(q_2)\,\dot{q}_2 & -0.084\sin(q_2)\,\dot{q}_2 \\ 0.084\sin(q_2)\,\dot{q}_1 & 0 \end{bmatrix}, \quad (3)$$

$$\boldsymbol{g}(\boldsymbol{q}) = 9.81 \begin{bmatrix} 3.921\sin(q_1) + 0.186\sin(q_1 + q_2) \\ 0.186\sin(q_1 + q_2) \end{bmatrix}, \quad (4)$$

$$\boldsymbol{f}(\dot{\boldsymbol{q}}) = \begin{bmatrix} 2.288\dot{q}_1 + 7.50\,\mathrm{sgn}(\dot{q}_1) \\ 0.175\dot{q}_2 + 1.734\,\mathrm{sgn}(\dot{q}_2) \end{bmatrix}. \quad (5)$$

4. CONTROL ALGORITHM AND MOTION SPECIFICATION

The primary goal of motion control in joint space is to make the robot joint positions $\boldsymbol{q}$ track a given time-varying desired joint position $\boldsymbol{q}_d$. Rigorously, the motion control objective in joint space is achieved provided that

$$\lim_{t \to \infty} \tilde{\boldsymbol{q}}(t) = \boldsymbol{0} \quad (6)$$

where $\tilde{\boldsymbol{q}}(t) = \boldsymbol{q}_d(t) - \boldsymbol{q}(t)$ denotes the joint position error.

A number of control schemes for achieving the motion control objective in joint space (6) have been reported in the literature. This paper concerns with the "computed torque control" which is a simple and natural appealing motion control scheme, whose practical effectiveness has been reported for more than two decades.

Let us introduce the following notation: Let $\boldsymbol{q}_d$, $\dot{\boldsymbol{q}}_d$, $\ddot{\boldsymbol{q}}_d$, be the desired joint position, velocity, and acceleration trajectories, which are chosen as bounded functions. The joint position and velocity errors are denoted by $\tilde{\boldsymbol{q}} = \boldsymbol{q}_d - \boldsymbol{q}$ and $\dot{\tilde{\boldsymbol{q}}} = \dot{\boldsymbol{q}}_d - \dot{\boldsymbol{q}}$, respectively, K_p and K_v are 2×2 symmetric positive definite matrices (Proportional and Derivative gain matrices respectively used in the controller).

Computed torque control is a special application of feedback linearization of nonlinear systems, which has been popular in modern systems theory; this is why this control technique is also called[6] "inverse dynamics control". This control scheme uses the robot dynamics in the feedback loop for linearization and decoupling. The computed torque controller is given by the following equation[6]

$$\boldsymbol{\tau} = M(\boldsymbol{q})[\ddot{\boldsymbol{q}}_d + K_v\dot{\tilde{\boldsymbol{q}}} + K_p\tilde{\boldsymbol{q}}] + C(\boldsymbol{q}, \dot{\boldsymbol{q}})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) + \boldsymbol{f}(\dot{\boldsymbol{q}}). \quad (7)$$

If the manipulator and friction dynamics models are exact, then the computed torque controller achieves dynamic decoupling of all the joints using nonlinear feedback, resulting in an asymptotically stable linear time-invariant error dynamics, and thus asymptotically exact tracking.

To show this, let us first compute the close-loop system by substituting the control action $\boldsymbol{\tau}$ from (7) into the right-hand side of the robot dynamics (1). This yields the closed-loop linear time-invariant differential equation:

$$\ddot{\tilde{\boldsymbol{q}}} + K_v\dot{\tilde{\boldsymbol{q}}} + K_p\tilde{\boldsymbol{q}} = \boldsymbol{0}$$

which can also be written in terms of the state vector $\begin{bmatrix} \tilde{\boldsymbol{q}}^T & \dot{\tilde{\boldsymbol{q}}}^T \end{bmatrix}$ as

$$\frac{d}{dt}\begin{bmatrix} \tilde{\boldsymbol{q}} \\ \dot{\tilde{\boldsymbol{q}}} \end{bmatrix} = \begin{bmatrix} 0 & I \\ -K_p & -K_v \end{bmatrix}\begin{bmatrix} \tilde{\boldsymbol{q}} \\ \dot{\tilde{\boldsymbol{q}}} \end{bmatrix}. \quad (8)$$

The unique equilibrium point of this system is the origin of the state space $\begin{bmatrix} \tilde{\boldsymbol{q}}^T & \dot{\tilde{\boldsymbol{q}}}^T \end{bmatrix}^T = \boldsymbol{0} \in \mathbb{R}^4$. The stability of this equilibrium can be concluded by invoking the Lyapunov's direct method (see Reference 9) by means of the following Lyapunov function[10]

$$V(\tilde{\boldsymbol{q}}, \dot{\tilde{\boldsymbol{q}}}) = \tfrac{1}{2}\begin{bmatrix} \tilde{\boldsymbol{q}} \\ \dot{\tilde{\boldsymbol{q}}} \end{bmatrix}^T \begin{bmatrix} K_p + \varepsilon K_v & \varepsilon I \\ \varepsilon I & I \end{bmatrix}\begin{bmatrix} \tilde{\boldsymbol{q}} \\ \dot{\tilde{\boldsymbol{q}}} \end{bmatrix} \quad (9)$$

where ε is any positive constant satisfying $\lambda_{\min}\{K_v\} > \varepsilon > 0$ with the symbol $\lambda_{\min}\{K_v\}$ denoting the smallest eigenvalue of the symmetric matrix K_v. This condition ensures that (9) is globally positive definite.

The time derivative along the trajectories of the closed-loop system (8) is given by

$$\dot{V}(\tilde{\boldsymbol{q}}, \dot{\tilde{\boldsymbol{q}}}) = -\begin{bmatrix} \tilde{\boldsymbol{q}} \\ \dot{\tilde{\boldsymbol{q}}} \end{bmatrix}^T \begin{bmatrix} \varepsilon K_p & 0 \\ 0 & K_v - \varepsilon I \end{bmatrix}\begin{bmatrix} \tilde{\boldsymbol{q}} \\ \dot{\tilde{\boldsymbol{q}}} \end{bmatrix}. \quad (10)$$

The global asymptotic stability is straightforward concluded from the Lyapunov's direct method because the Lyapunov function (9) is globally positive definite whereas its time derivative (10) is globally negative definite. One consequence of this result is that the state vector composed by $\dot{\tilde{\boldsymbol{q}}}(t)$ and $\tilde{\boldsymbol{q}}(t)$ vanishes as $t \to \infty$. This implies that the motion control objective (6) is guaranteed.

In practice, however, the presence of unmodeled high-frequency dynamics and noise in the velocity estimation causes deviations in trajectory tracking, so the best we can expect is ultimated boundedness of the state vector.

The control law (7) has been coded in *C* language considering the gain matrices K_p and K_v as diagonal, whose entries are denoted by k_{p1}, k_{p2}, and k_{v1}, k_{v2}, respectively. Although the closed-loop stability is guaranteed for all positive value of the gains, in practice the values of these parameters affect the control system performance. The user of our system can corroborate – from position error plots – the effect of changing these parameters from the GUI, where they are restricted to the following intervals:

$k_{p1} \in$ (200.0, 600.0) [Nm/rad],
$k_{p2} \in$ (600.0, 1500.0) [Nm/rad],
$k_{v1} \in$ (20.0, 60.0) [Nm sec/rad],
$k_{v2} \in$ (40.0, 100.0) [Nm sec/rad].

https://doi.org/10.1017/S0263574700003167 Published online by Cambridge University Press

4.1. Motion specification

An important constituent for experimental evaluation of robot control algorithms is the choice of a class of desired trajectories $\boldsymbol{q}_d(t)$ for the robot motion. The desired trajectory should be chosen to exhibit motion profile without abrupt changes in position, velocity and acceleration from the beginning to the end of the motion, while preventing the actuators from saturating. The expression of the desired trajectory used in this project is similar to those proposed in References 11 and 12, that is

$$\boldsymbol{q}_d(t) = \begin{bmatrix} 45.0[1 - e^{-2.0\,t^3}] + a_1[1 - e^{-2.0\,t^3}] \sin(\omega_1 t) \\ 60.0[1 - e^{-1.8\,t^3}] + a_2[1 - e^{-1.8\,t^3}] \sin(\omega_2 t) \end{bmatrix} \text{ [deg]} \tag{11}$$

where a_1 and a_2 denote the amplitude of the steady state sine functions whereas ω_1 and ω_2 represent the frequency of the desired trajectory for the shoulder and elbow joints respectively.

The user can modify the value of amplitude and frequency of the desired motion directly from the graphical interface within the following intervals:

$$a_1 \in (7.0, 15.0) \text{ [deg]},$$
$$a_2 \in (100.0, 150.0) \text{ [deg]},$$
$$\omega_1 \in (0.0, 20.0) \text{ [rad/sec]},$$
$$\omega_2 \in (0.0, 5.0) \text{ [rad/sec]}.$$

The reason for the choice of this trajectory structure (11) was twofold. First, at the beginning of the tests – time t = 0 [sec] – the desired joint positions $\boldsymbol{q}_d$, velocities $\dot{\boldsymbol{q}}_d$ and accelerations $\ddot{\boldsymbol{q}}_d$ are zero. Since the arm starts at rest, then the control system will present nice smooth transient. Second, evaluation of the robot dynamics (1) along the desired trajectory offers a good idea about the "ideal" torque inputs needed to match the desired motion. For this trajectory, the "ideal" torques are around 85% of the motor capabilities — 200 [Nm] and 15 [Nm], respectively — .

5. INTERFACES

An informative yet simple graphic interface is essential for a robot control system to be productive and easy to use. The challenge is to provide enough information to make the interpretation easy while minimizing data transmission and maintaining a simple layout.

Good interface design is largely an iterative process, that can be simplified by the use of suitable computing tools. We used the Graphics Editor of the TILCON Real-Time Developer, which is a tool for designing object-oriented and event-driven GUIs.[4] Although only one interface is required in the system, we have decided to develop two interfaces. Figures 5 and 6 show these interfaces.

The first interface, depicted in Figure 5, is composed by two sections. The upper section is devoted to monitor the joint position errors $\tilde{q}_1$ and $\tilde{q}_2$ by plotting their time evolution. Also, this section contains three icons for energizing the robot actuators, starting and finishing the control algorithm, and quitting the whole system.

The lower section is divided into four panels. The "Torques" and "Velocities" panels display the corresponding signals τ_1, τ_2 and $\dot{q}_1$, $\dot{q}_2$, from the robot controller. The other two panels, "Controller gains" and 'Trajectory parameters", let the user modify the controller gains k_{p1}, k_{p2}, k_{v1}, and k_{v2}, as well as the desired motion amplitudes and frequencies a_1, a_2, and ω_1, ω_2, respectively. This is

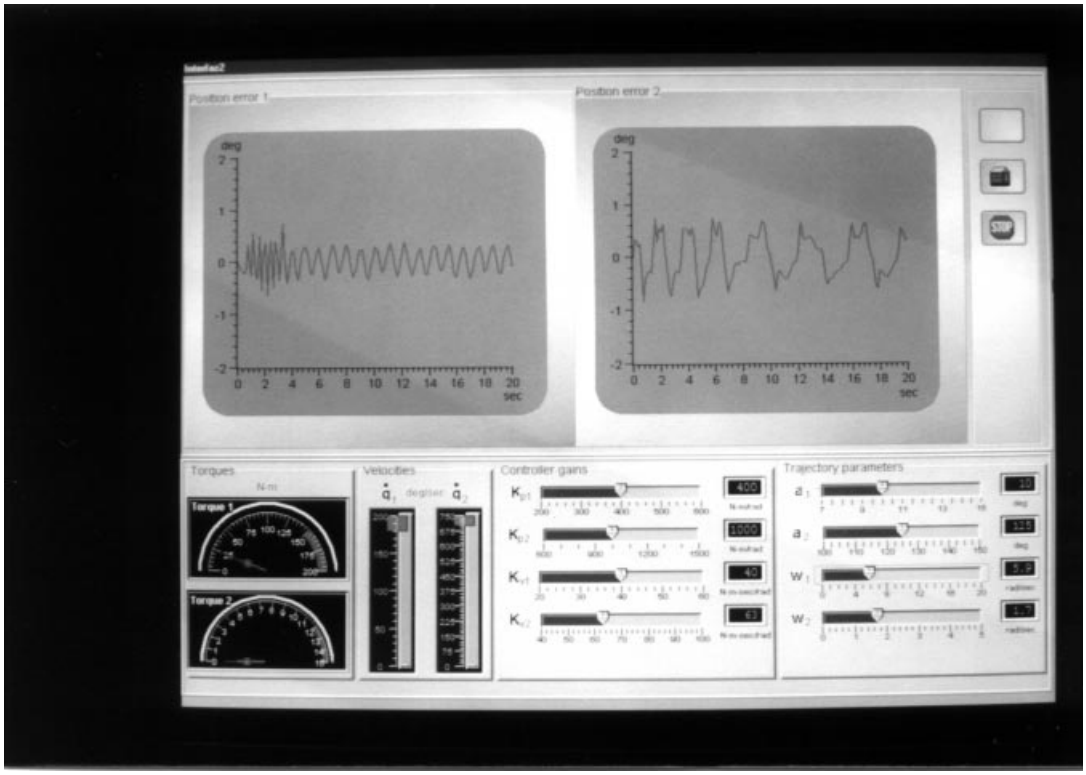

Fig. 5. Two-charts Graphical user interface.

https://doi.org/10.1017/S0263574700003167 Published online by Cambridge University Press

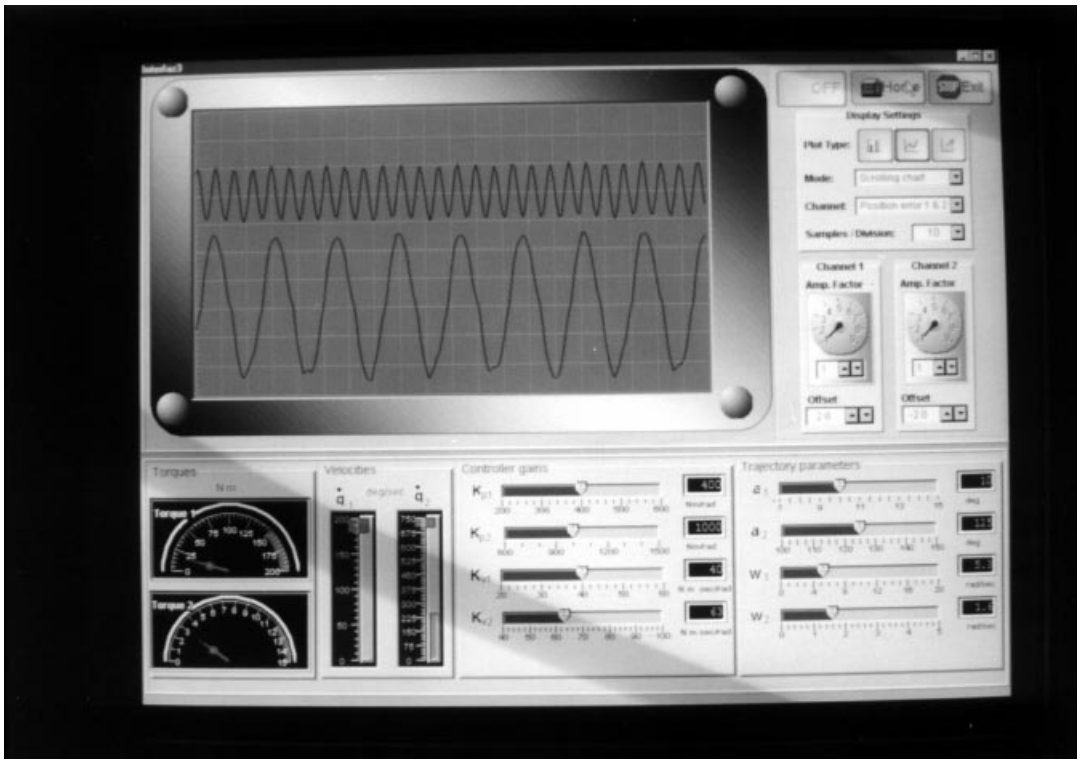

Fig. 6. Oscilloscope-like graphical user interface.

performed either by using sliding bars or directly from the keyboard.

The second interface, shown in Figure 6, is also composed by two sections. The main difference with respect to the first interface in Figure 5 is that in the upper section the two plots have been replaced by an oscilloscope-like screen with two input channels. The functionality of this oscilloscope is set through the upper right panel. This panel contains different options that let the user specify the display settings, such as the plot type (discrete dots, continuous line of filled curve), the mode (scrolling or stripping chart), the channel to display (either $\tilde{q}_1$, $\tilde{q}_2$, or both), or the number of points per division of the screen.

The usage of both interfaces is far simple, so letting the users the opportunity to corroborate in an easy way, some important facts about robot motion control. For example, the user can vary the parameters of the controller and check, among other things, that:

- High proportional gain k_p reduces the tracking position error $\tilde{q}$ but produces oscillations and actuator saturations.
- Small derivative gain k_v leads to oscillations. A good choice for k_v is to respect the critical damping condition[6] $k_v = 2\sqrt{k_p}$.
- High motion amplitude a increases the tracking error $\tilde{q}$ and may cause actuator saturation.
- High motion frequency ω also increases the tracking error $\tilde{q}$ and may cause actuator saturation.

6. CONCLUDING REMARKS

Most of present automation systems are based on real-time control systems implemented in digital computers. 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 a single-processor computer.

The main features of real-time systems, such as concurrent processes being executed at desired rates and interchanging data, have been illustrated in this paper considering one of the applications where timing requirements are critical: robotics.

A real-time control system has been developed on a single-processor PC based on Windows NT's real-time extension RTX and using a graphical user interface built with the TILCON Real-Time Graphics Editor. The controlled real-world system was a mechanical arm formed by two joints. The control algorithm selected to handle the arm was the so-called "computed torque control" which was coded in *C* language and executed every 2.5 [msec] thanks to the deterministic responses of the RTX module.

Besides the concepts of real-time control systems, the developed application also allows the user – through the friendly user interfaces – to learn about manipulator control facts such as the effect in motion tracking accuracy of changing in the proportional and derivative gains, and the desired motion amplitudes and frequencies.

References

1. M. Brady, "Editorial: Preface to the millennium special issue", *Int. J. Robotics Research* **18**, No. 11, 1051–1055 (November, 1999).
2. K. Arzen, "Real-time systems", Lecture 1: Engineering course; http://control.1th.se/kurstr/material99.html.
3. VenturCom Inc., *RTX 4.3 User's Guide* (VenturCom, Inc., Cambridge, MA, 1999).
4. Tilcon Software Ltd., *TILCON Real-Time Developer V4*; http://www.tilcon.com (Ontario, Canada).
5. H. Asada and K. Youcef-Toumi, *Direct-drive Robots* (The MIT Press, 1987).

https://doi.org/10.1017/S0263574700003167 Published online by Cambridge University Press

6. M.W. Spong and M. Vidyasagar, *Robot Dynamics and Control* (John Wiley and Sons, New York, 1989).
7. N. Costescu, D. Dawson and M. Loffler, "Qmotor 2,0 – A Real-time PC based control environment", *IEEE Control Systems* **19**, No. 3, 68–76 (June, 1999).
8. F. Reyes and R. Kelly, "Experimental evaluation of identification schemes on a direct drive robot", *Robotica* **15**, Part 5, 563–571 (September-October, 1997).
9. M. Vidyasagar, *Nonlinear Systems Analysis* (Prentice Hall, 1993).
10. J.T. Wen and D.S. Bayard, "New class of control laws for robotic manipulators. Part 1: Non-adaptive case", *Int. J. Control* **47**, No. 5, 1361–1385 (1988).
11. D.M. Dawson, J.J. Carroll and M. Schneider, "Integrator backstepping control of a brush dc motor turning a robotic lead", *IEEE Transactions on Control System Technology* **2**, No. 3, 233–244 (September, 1994).
12. M.S. de Queiroz, D. Dawson and T. Burg, "Reexamination of the DCAL controller for rigid link robots", *Robotica* **14**, Part 1, 41–49 (1996).

https://doi.org/10.1017/S0263574700003167 Published online by Cambridge University Press