A One-Stop Solution in Robotic Control System Design

by
S.S. GE,
T.H. LEE,
D.L. GU,
and
L.C. WOON

OpenRob: An Open-Architecture Platform for Model Building, Controller Design, and Numerical Simulation

1070-9932/99/$10.00©2000IEEE

To date, there are already some commercial CAD software for robotics, such as Workspace [1], Robotica [2,3] and the Robotics Toolbox for MATLAB. For both Robotica and the Robotics Toolbox of MATLAB, only a collection of functions is provided to be used in either the Mathematica or MATLAB environment. Although Workspace provides a powerful visualization tool of mechanical design for robots, it does not have the functionality to integrate trajectory planning, dynamic simulation, and controller design together.

The current practice of controller design for a given robot is very troublesome. Engineers and researchers normally need to use a combination of tools for model building, controller design, and numerical simulation. Additional efforts are necessary in converting the controller algorithms in higher-level languages into real-time control codes. Though some of the software can be converted into C automatically for real-time implementation, the resulting code is generally very large and not optimized. During this process, many problems are encountered because design and implementation are developed in two or more different environments. The development life cycle is thus very long.

All these factors have motivated the design and development of a new integrated open-architecture platform for robot simulation, controller design, and real-time implementation—OpenRob. The proposed system provides not only a user-friendly simulation and educational platform but also an open-architecture, control system development environment for real-time control of robotic systems in conjunction with an appropriate motion-controller card. Besides the built-in conventional robot models, trajectory planning, and motion-control algorithms, OpenRob also enables the user to integrate user specified robots, control algorithms, and trajectory-planning schemes into the system. This open system provides a one-stop solution in the sense that model building, controller design, and numerical simulation are integrated into one platform for the ease of real-time implementation. This will greatly shorten the development life cycle and avoid the problem of compatibility and portability of codes. In addition, the software can also be used as a tool for teaching robotics courses. In this article, we present an overview of the OpenRob software, discuss implementation issues, and provide a few case studies involving OpenRob.

Overview of OpenRob

OpenRob is developed using the standard style of Windows application that can be run on any Win32 platform: Windows 95/98/NT. Figure 1 shows the main window platform of OpenRob. It is divided into two panes. The upper one is the `System Configurator` pane in which all the configuration settings are managed while the lower one is the `Performance Graphs` pane that can be used to display the control performance graphically.

A friendly and interactive user interface is provided in the `System Configurator`, which is divided into three functional modules to represent three main system functions: `Trajectory Planner`, `Controller`, and `Robot`. For each module, there are two combo list boxes for the user to configure the corresponding category and type. By left-clicking the mouse on the modules, the corresponding dialog box or property sheet will be popped out for setting the configuration of the module. After each functional module has been configured, the `Motion Commander` button on the toolbar is pushed to load the specified trajectory planning, motion-control algorithm, and the corresponding robot model into the system for simulation study. The tracking performances are displayed as graphs in the lower `Performance Graphs` panes.

OpenRob enables a user to customize his own robot and build it into the system for simulation and real-time control.

Furthermore, OpenRob also provides a user-friendly interface for combining customized robot models, trajectory planning, and control algorithms into the system conveniently. This is done by choosing the `User Defined` type item in the combo list box for each function module, and an open file dialog box will be popped out for user to select the customized dynamic link libraries (DLLs) from the corresponding directory. As long as the DLL designed by the user complies with the well-defined export function declaration, it will be automatically loaded and executed.

The main features of this open-architecture platform include:

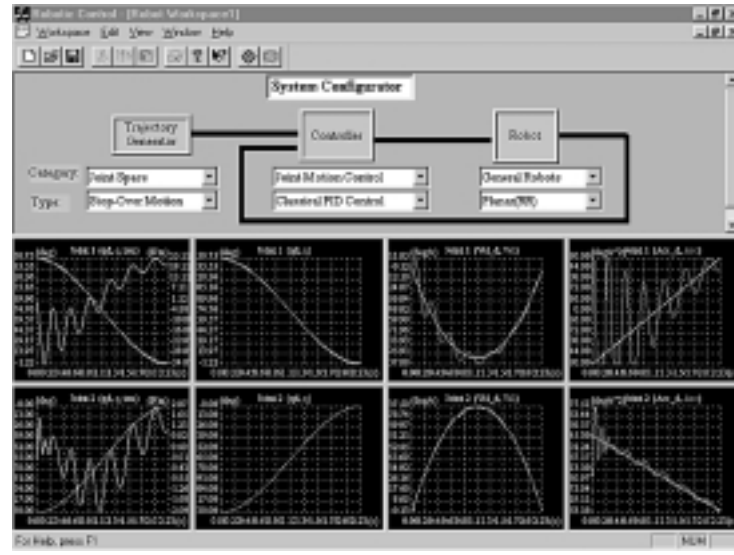

Figure 1. *Sample layout of the system platform.*

- Several workspaces can be opened and executed at the same time.
- A powerful `System Configurator` is provided for easy configuration and friendly man-machine communication.
- A flexible graphic plotting environment is embedded in the system for easy monitoring and observing of the control performance.
- Eight commonly used robots, including some commercial robots, have been built into the system, which include their forward/inverse kinematics and dynamic models [4-6].
- Several common trajectory-planning algorithms have been introduced into the system, such as Linear Function with Parabolic Blend, Cubic Polynomial, Quintic Polynomial, and the like, in which the optimal nonlinear programming schemes are also integrated [7-9].
- Six different motion-control algorithms have been implemented in the system, which includes classical PID control, variant PID control, computed torque control, sliding-mode control, adaptive neural network control, and adaptive structural network control [10-13].
- An open system interface is provided for the user to load the custom robot model, trajectory-planning algorithms, and control schemes into the system.

```
#A ROBOT input data file for " planar.cfg "

DOF                 = 2
Joint Type = RR

The Denavit-Hartenberg parameters:
a[1]                = 1.000000
alpha[1]            = 0.000000
d[1]                = 0.000000
theta[1]            = 0.000000

a[2]                = 1.000000
alpha[2]            = 0.000000
d[2]                = 0.000000
theta[2]            = 0.000000

The link inertia parameters:
m[1]                = 1.000000
masscenter[1]       = -0.500000,0.000000,0.000000
I[1]                = 0.000000,0.333333,0.333333,0.000000,0.000000,0.000000

m[2]                = 1.000000
masscenter[2]       = -0.500000,0.000000,0.000000
I[2]                = 0.000000,0.333333,0.333333, 0.000000,0.000000,0.000000

The Joint Limits parameters:
PosLimitLo[1]       = -3.141593
PosLimitHi[1]       = 3.141593
MaxA[1]             = 3.141593
MaxV[1]             = 3.141593
MaxTau[1]           = 1.300000

PosLimitLo[2]       = -3.141593
PosLimitHi[2]       = 3.141593
MaxA[2]             = 6.283185
MaxV[2]             = 6.283185
MaxTau[2]           = 2.300000

The Dynamics parameters:
P_num               = 5
P                   = 2.333000,0.333300,0.500000,14.710080,0.000000,
```

Figure 2. *A sample configuration file for a planar robot.*

Implementation Issues

In order to make the system more flexible, portable, and a "true" open-architecture platform, the DLL technology has been incorporated into the system. The DLL is an executable file that acts as a shared library of functions. Dynamic linking provides a way for a process to call a function that is not part of its executable code. The executable code for the function is located in a DLL, which contains one or more functions that are compiled, linked, and stored separately from the processes that use them. DLLs also facilitate the sharing of data and resources. Multiple applications can simultaneously access the contents of a single copy of a DLL in memory. The optional DLL packages in OpenRob have been divided into three categories in accordance with the three main functional modules. These built-in DLLs will be loaded automatically when the specific functional categories and types are selected.

Robot Module

KINEMATICS

For the ease of maintenance and updating of the robot configuration data, a special configuration file (.cfg file) is designed for each type of robot to store the default configuration declared in data structure `Robot.` Hence, when a particular type of robot is chosen, OpenRob will load the default .cfg file into the memory and reinitiate the `Robot` structure automatically. The format of the .cfg file is defined as follows, and a sample configuration file for a planar two-link robot is shown in Fig. 2.

1. The first line of the .cfg should be a comment line beginning with symbol "#."

2. Each configuration data occupies one line and it should be prefixed with symbol "=." Before symbol "=" any characters can be added. Empty lines are allowed between two configuration data items.

3. The sequence of the configuration data is as follows:

(a) Number of degrees of freedom;

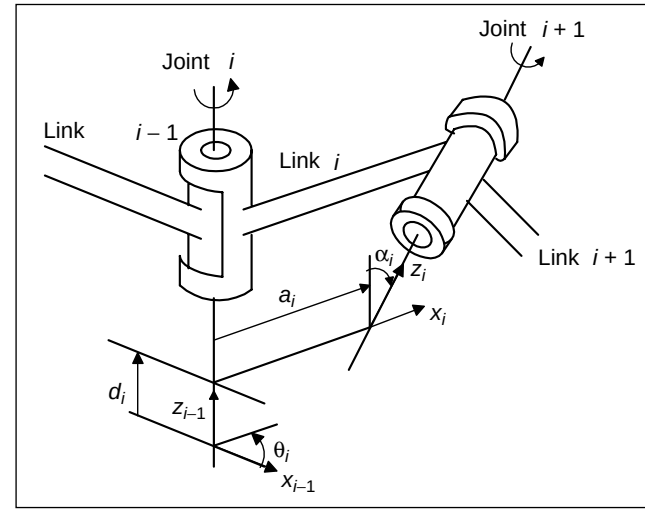

Figure 3. *Definition of the link and joint parameters.*

(b) Sequence of joint types denoted with a string of "P" and "R," in which "P" is used to represent a prismatic joint while "R" represents a revolute one.

(c) D-H representation parameters a, α, d, θ for each DOF (refer to Fig. 3).

(d) Link inertia parameters such as mass, mass center, and inertia tensor for each link.

(e) Joint limits parameters such as low and high position limits, velocity, acceleration, and torque/force limits.

(f) The dimension and elements of the parametric vector for the linear-in-the-parameters (LIPs) expression of the dynamics [5, 13].

To implement the basic operations in kinematics, six functions are provided as follows:

- `PToCylindric()`: Converts the position description from Cartesian coordinates to cylindrical coordinates.
- `PToSpheric()`: Converts the position description from Cartesian coordinates to spherical coordinates.
- `CartToEuler()`: Converts the orientation presentation from Cartesian coordinates to Euler angles (roll, pitch, yaw).
- `FrameTransform()`: Calculates the numerical result of the transformation matrix T_j^i from frame j to frame i for a given robot configuration and joint variables.
- `ForwardKinematics()`: Calculates the numerical result of the transformation matrix T_n^0 from the end-effector frame to the base frame for a given robot configuration and joint variables.

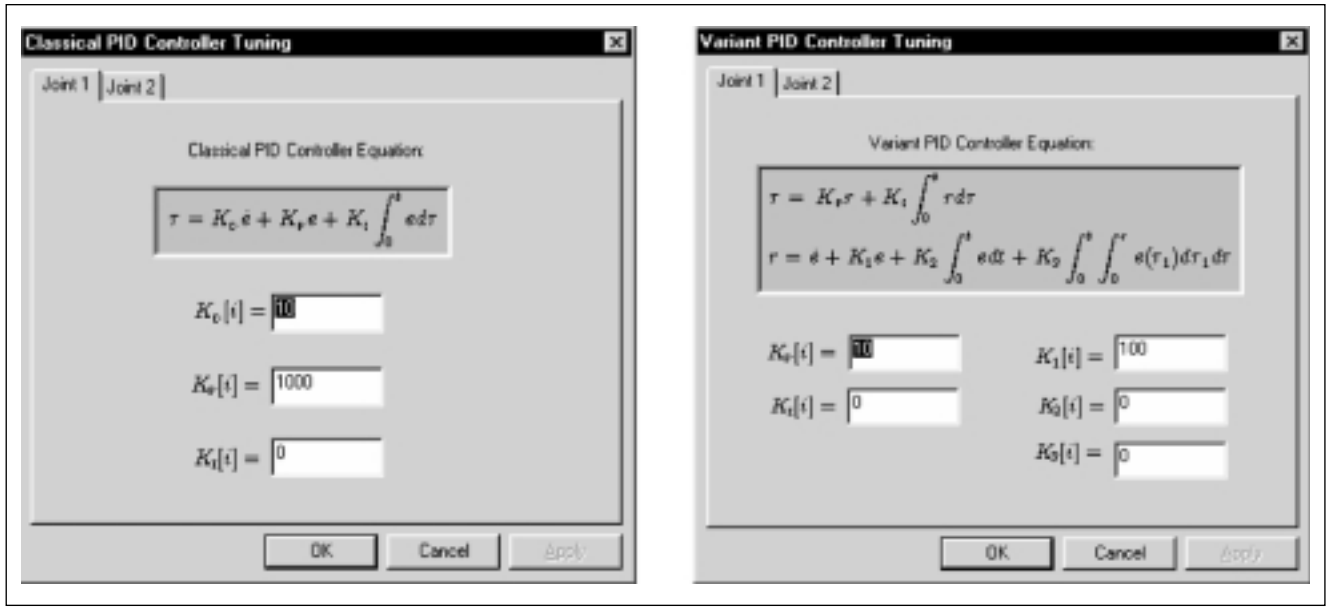

Figure 4. *Tuning pages for classical (left) and variant (right) PID controllers.*

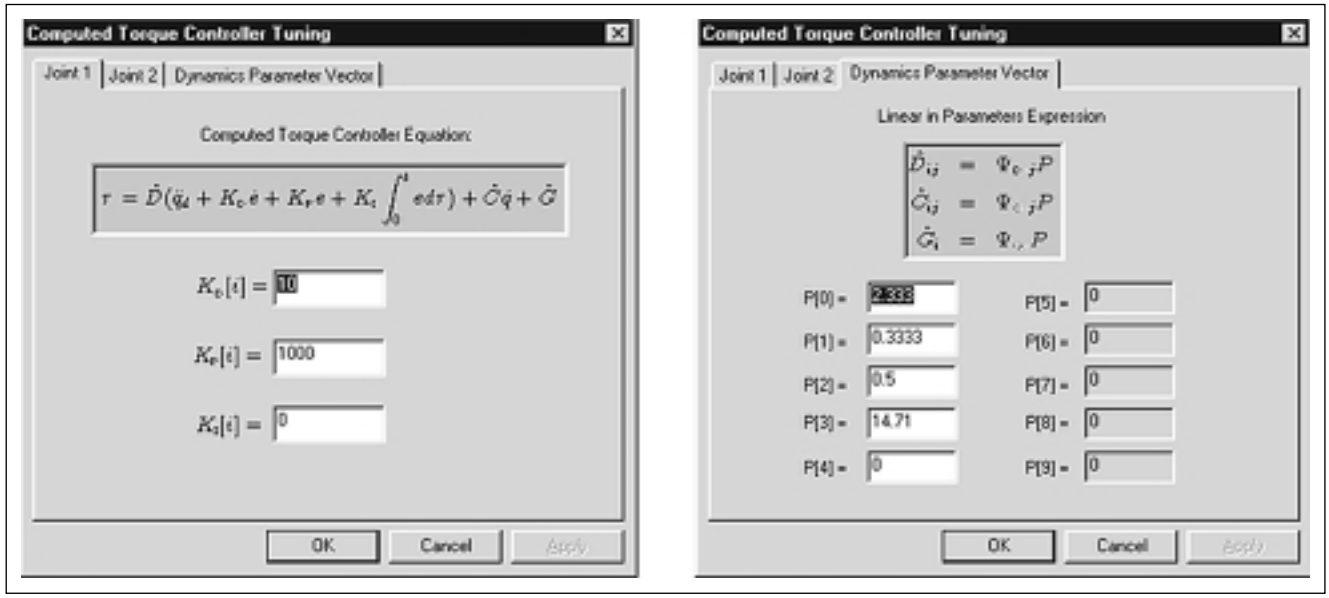

Figure 5. *Tuning pages for a computed torque controller.*

- `ForwardJacobian()`: Calculates the linear and angular velocities of the end-effector for a given robot configuration and joint velocities by using the Jacobian matrix.

Given the robot configuration and joint variables, the function `FrameTransform()` can be used to generate any transformation matrix T_i^j where $i \geq j$, $i, j = 0, \ldots, n$. Therefore, if we set `initial` equal to 0 while `final` is equal to the DOF of the robot, then the matrix obtained will be the forward kinematics T. Furthermore, the position and orientation of the end-effector given in T can also be converted to other representation forms using the functions `PToCylindric()`, `PToSpheric()`, and `CarToEuler()`.

According to the presentation of the desired path, trajectory planning can be conducted either in joint space or in Cartesian space.

With the `FrameTransform()`, the forward Jacobian can be generated easily. By calculating T_i^0 for $i = 0, \ldots, n$, and extracting the first three elements in the third and fourth columns of T_i^0 as Z_i and O_i, respectively, the Jacobian J can be constructed using the following equations [10]:

$$J = [J_1 \quad J_2 \quad \cdots \quad J_n] \tag{1}$$

where the ith column J_i is given by

$$J_i = \begin{cases} \begin{bmatrix} Z_{i-1} \times (O_n - O_{i-1}) \\ Z_{i-1} \end{bmatrix}, & \text{if joint } i \text{ is revolute} \\ \begin{bmatrix} Z_{i-1} \\ 0 \end{bmatrix}, & \text{if joint } i \text{ is prismatic.} \end{cases} \tag{2}$$

Besides the above algorithms, a graphical user interface (GUI) property sheet for the `Robot` module is built-in in this open-architecture system for easy understanding and operation. By left-clicking the `Robot` module box in the `System Configurator`, the robot property sheet pops out with six property pages: `Link Parameters`, `Inertia Parameters`, `Robot Limits`, `Kinematics`, `Jacobian`, `Dynamics`. In the `Link Parameters`, `Inertia Parameters`, and `Robot Limits` property pages, by increasing or decreasing the joint index, the configuration data for each joint is displayed. The configuration data can also be changed and saved to file except for the commercial robots such as Puma and Stanford. The `Kinematics` and `Jacobian` property pages pro-

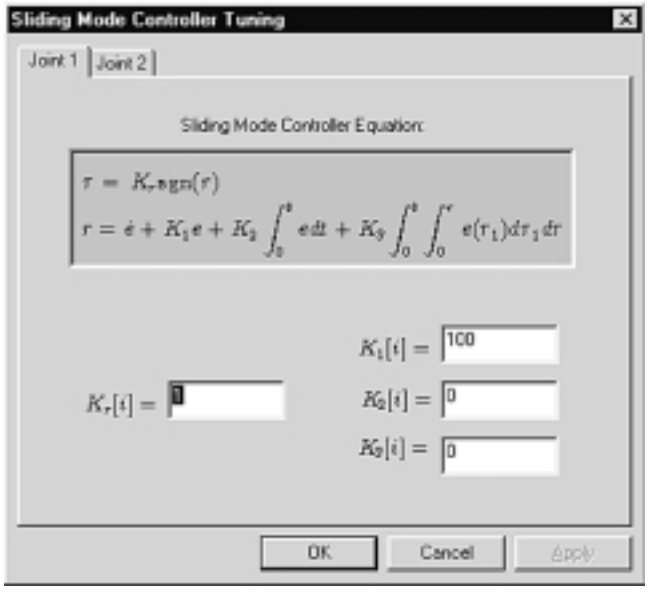

Figure 6. *Tuning page for a sliding-mode controller.*

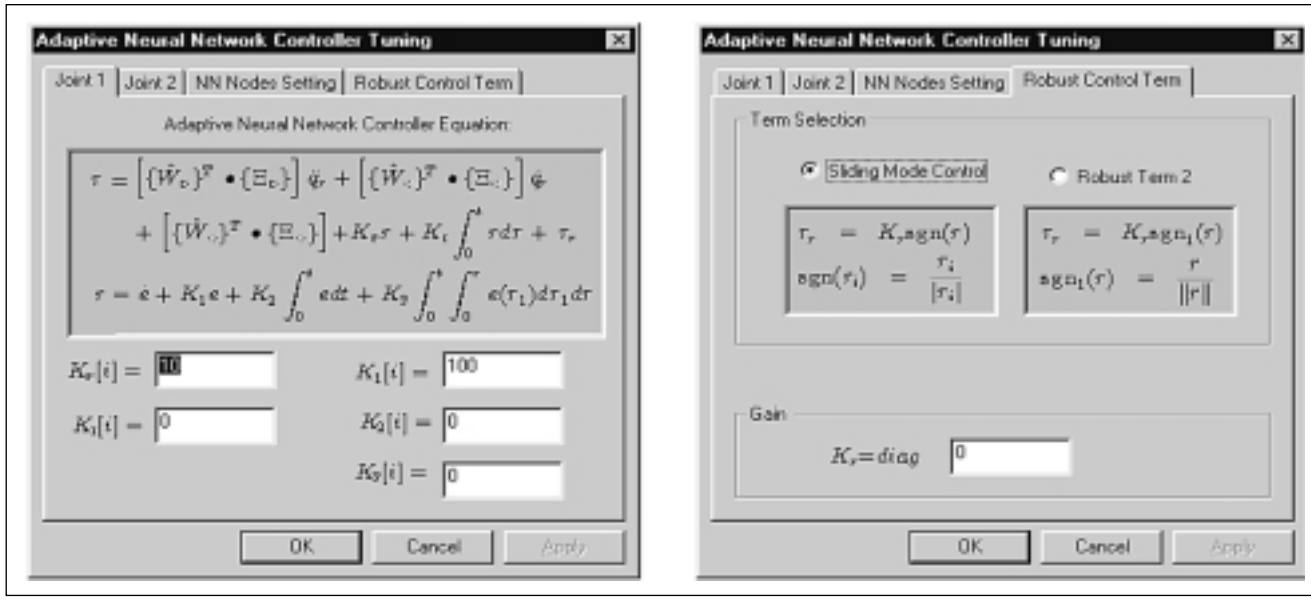

Figure 7. *Tuning pages for an adaptive neural network controller.*

vide a learning tool to study kinematics and Jacobian properties of robots. After keying in the joint positions for each joint and clicking the `Forward Kinematics` button, the position and orientation of the end-effector can be obtained and displayed in the lower frames. Furthermore, different descriptions for position in Cartesian, cylindrical, and spherical spaces and for orientation in transformation matrix or Euler angle representations can also be selected from the combo list box and the displayed data will be automatically updated accordingly. In the same way, after filling in the joint position and velocities, the linear and angular velocities of the end-effector can also be obtained in the `Jacobian` property page.

DYNAMICS

From Lagrangian-Euler formulation, the dynamic equations of an n link manipulator can be stated as

$$D(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau \qquad (3)$$

where $q \in R^n$ is the vector of joint variables, $\tau \in R^n$ is the vector of joint torque applied by the actuators, $D(q) \in R^{n\times n}$ is the symmetric positive definite inertial matrix, $C(q,\dot{q}) \in R^{n\times n}$ is the Coriolis and centrifugal force matrix, and $G(q) \in R^n$ is the gravitational force vector [5, 12, 13].

The dynamic equations are usually used in the computer simulation of robot motion and controller design. The following three basic algorithms are provided in OpenRob:

- `DynamicsSimulation()`: Given the current states (joint position and velocity) and the torque of the robot, the Runge-Kutta-Merson algorithm is called to solve the differential equations to obtain the next states of joint position and velocity.
- `SolveODE()`: The robust adaptive step size numerical algorithm of the Runge-Kutta-Merson method is used to solve the ordinary differential equations [13].
- `AutoDeriveDynamics()`: Given the current state variable $x = [q^T \ \dot{q}^T]^T$, input force/torque, and robot configuration parameters, $\dot{x}$ is calculated as the result of the function.

As an open-architecture platform, OpenRob enables a user to customize his own robot and build it into the system for simulation and real-time control. A user interface has been designed to load the custom robot into the system. If the selected DLL file and corresponding .cfg file can be loaded successfully, the DLL file name will be updated in the robot type combo list box and the system will be adjusted automatically to this custom environment so that all the existing tools in the system can be used to access this custom robot.

Trajectory-Planning Module

Trajectory planning is another important module in the control of robotic manipulators. It can reduce a user's efforts in specifying the complicated functions of space and time. Given the desired destination location or the traveling path with a sequence of via points, the trajectory planner generally interpolates it by a class of polynomial functions and generates a sequence of time-based joint position, velocity, and acceleration.

According to the presentation of the desired path, trajectory planning can be conducted either in joint space or in Cartesian space. For joint-space planning, the time history of all joint variables and their first and second derivatives are planned to describe the desired motion of the manipulator. For Cartesian-space planning, the time history of the end-effectors' position, velocity, and acceleration are planned first, and the corresponding joint position, velocity, and acceleration are then derived from inverse kinematics.

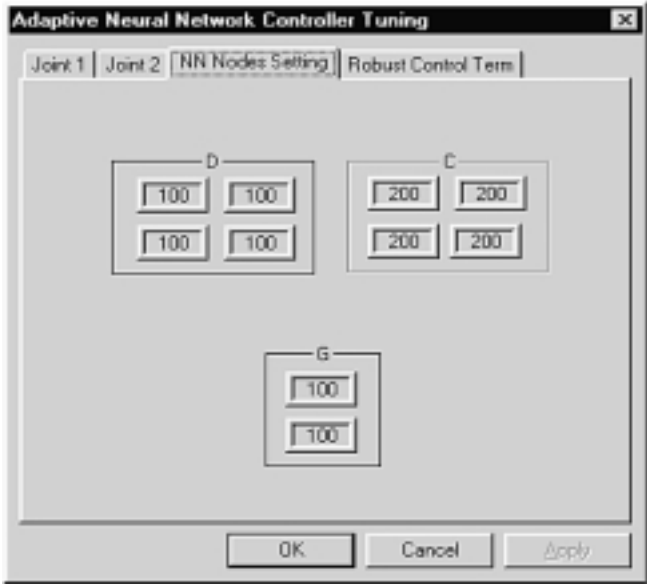

Figure 8. *Property page for a neural network node distribution.*

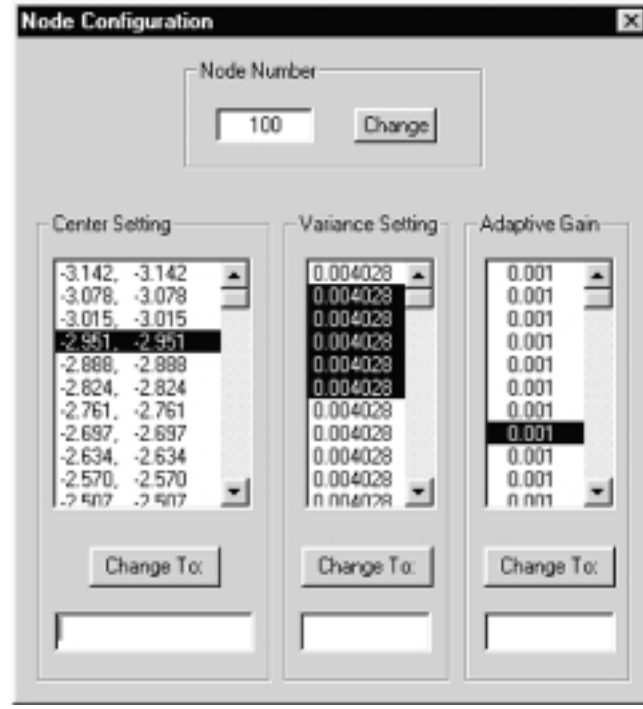

Figure 9. *Neural network node configuration dialogue box.*

Normally, motion can be divided into two types. The first one is stop-over motion in which the robot stops at each desired via point, while the other one is continuous motion in which the robot goes through each via point without stopping.

STOP-OVER TRAJECTORY PLANNING

There are two types of planning methods to specify the stop-over motion: synchronized and unsynchronized schemes. In synchronized motion all joints arrive at each via point simultaneously, while in unsynchronized motion all joints reach their destinations as fast as possible according to their own dynamics. The common requirement is that the minimum time interval for each joint under its own velocity and acceleration constraints during each motion section should be determined first. Then, in synchronized motion, the longest time among them will be selected as the common time interval so that all the joints can finish their motion simultaneously; and in unsynchronized motion, each joint selects its own minimum travel time to reach the via point.

In general, polynomial interpolation is widely used to design the trajectory. In order to create a smooth path with continuous position and velocity, we start with the linear function

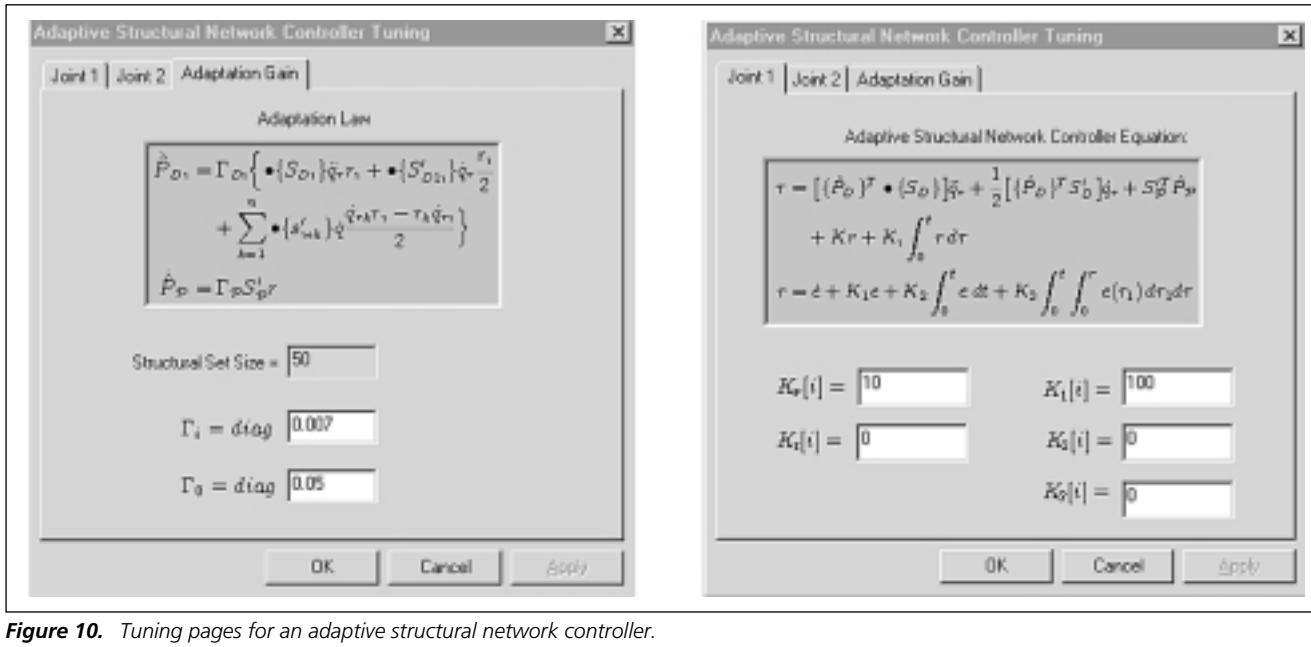

Figure 10. *Tuning pages for an adaptive structural network controller.*

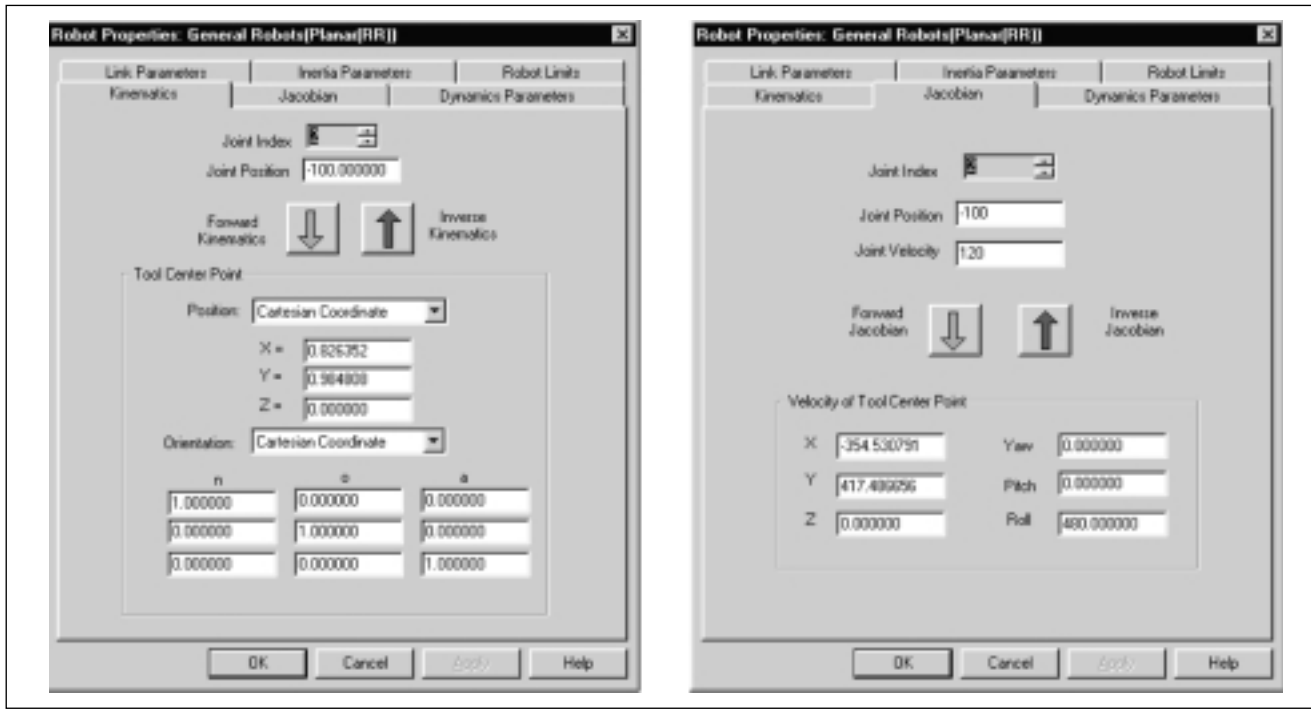

Figure 11. *Screen shots for a planar robot's kinematics and dynamics properties.*

Table 1. Control Algorithms Implemented in OpenRob		
Control Type	**Control Law**	**Adjustable Gains**
Classical PID	$\tau = K_D\dot{e} + K_P e + K_I\int_0^t e\, d\tau$	K_D, K_P, and K_I
Variant PID	$\tau = K_P r + K_I\int_0^t r d\tau$	K_P, K_I, K_1, K_2, and K_3
Computed Torque	$\tau = \hat{D}(q)(\ddot{q}_d + K_D\dot{e} + K_P e + K_I\int_0^t e\, d\tau) + \hat{C}(q,\dot{q})\dot{q} + \hat{G}(q)$	K_D, K_P, and K_I
Sliding Mode	$\tau = K_r \mathrm{sgn}(r)$	K_r, K_1, K_2, and K_3
Adaptive NN [13]	$\tau = [\{\hat{W}_D\}^T \bullet \{\Xi_D\}]\ddot{q}_r + [\{\hat{W}_C\}^T \bullet \{\Xi_C\}]\dot{q}_r + [\{\hat{W}_G\}^T \bullet \{\Xi_G\}] + K_P r + K_I\int_0^t r d\tau + \tau_r$	K_P, K_I, K_1, K_2, K_3, and K_r
Adaptive SN [13]	$\tau = [\{\hat{P}\}^T \bullet \{S\}]\ddot{q}_r + \frac{1}{2}[\{\hat{P}\}^T W_s]\dot{q}_r + W_0^T\hat{P}_0 + K_P r + K_I\int_0^t r d\tau$	K_P, K_I, K_1, K_2, and K_3

but add a blend region at each path point. Different acceleration and velocity profiles can be selected along the trajectory, such as linear function with parabolic blends (LFPB), linear function with cubic blends (LFCB), and linear function with quartic blends (LFQB). Sometimes, high-order polynomial functions, such as cubic and quintic polynomial interpolations, are used for the smoothness of the signals.

CONTINUOUS CUBIC OPTIMAL TRAJECTORY PLANNING

In continuous go-through motion, let N be the number of the desired via points that include the start and end points, q_{ji} be the position of joint j at the via point i, and t_i be the time when the robot passes through the via point i for $j = 0, \ldots, n-1$ and $i = 0, \ldots, N-1$. The piecewise cubic function is used for joint j between q_{ji} and $q_{j(i+1)}$.

In order to maximize the speed of operation, the traveling time for the robot should be minimized. In OpenRob, Nelder and Mead's flexible polyhedron search [8] has been utilized to accomplish this.

DESIGN OF TRAJECTORY PLANNER

In OpenRob, `Trajectory Planner` is designed to configure and implement the mentioned trajectory-planning algorithms as individual DLLs. As an open-architecture platform,

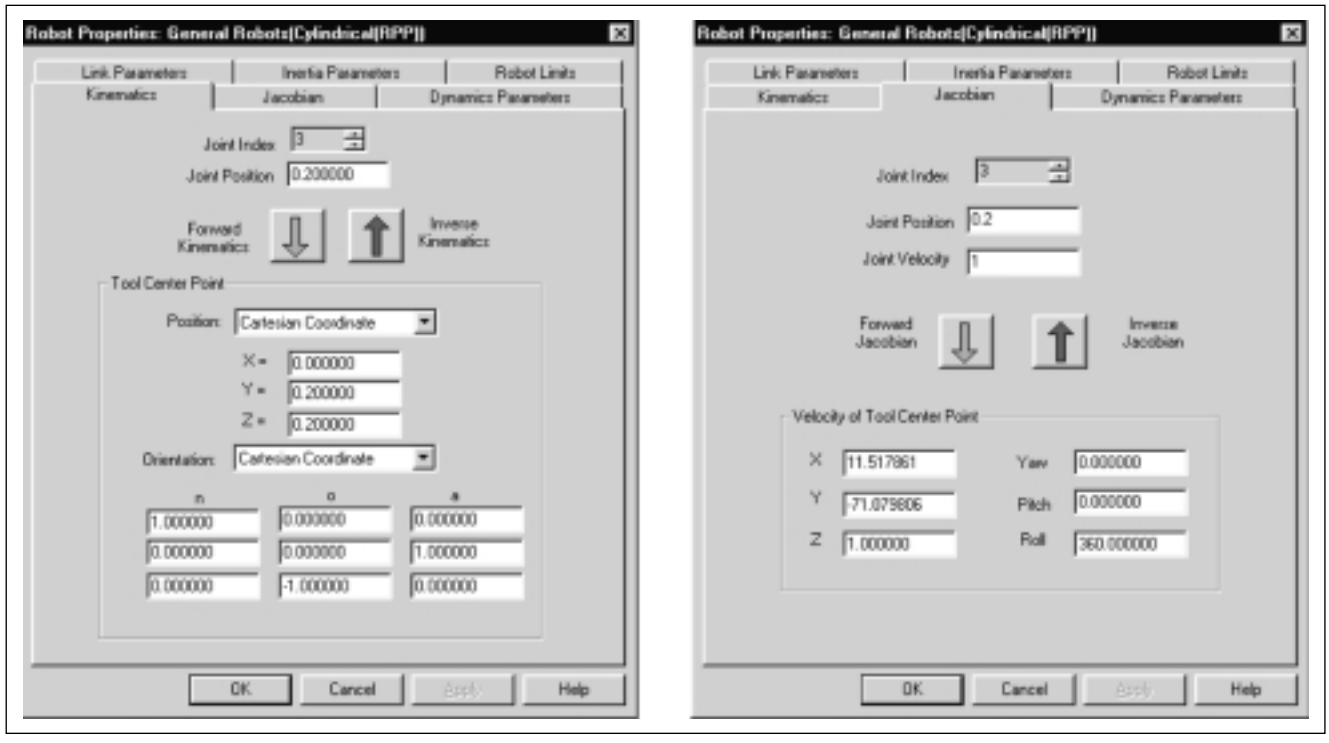

Figure 12. *Screen shots for a cylindrical robot's kinematics and dynamics properties.*

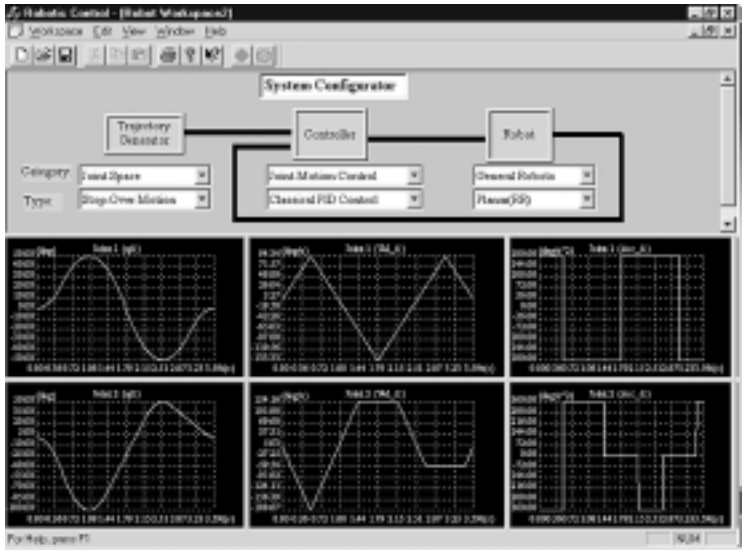

Figure 13. *Trajectory planned with the LFPB method in stop-over motion (Case 2).*

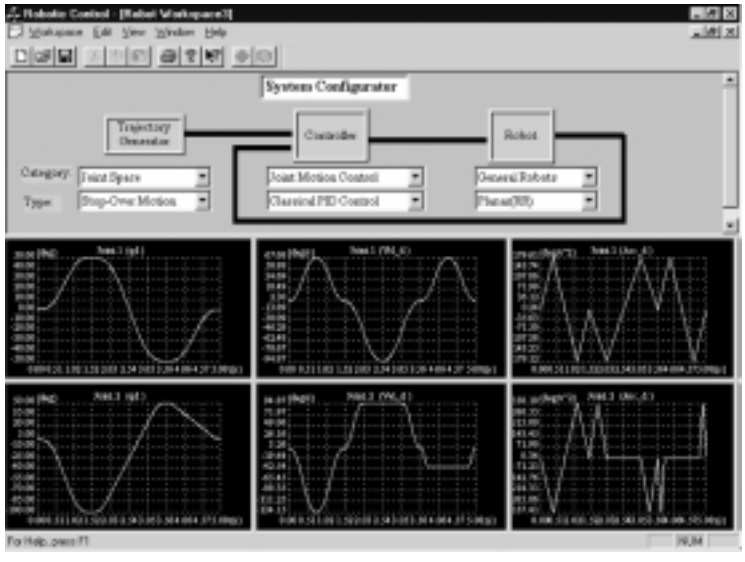

Figure 14. *Trajectory planned with the LFCB method in stop-over motion (Case 2).*

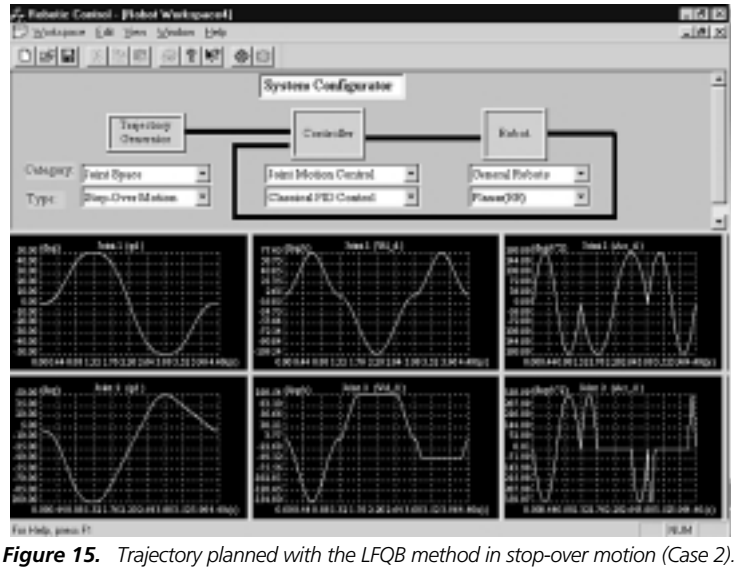

Figure 15. *Trajectory planned with the LFQB method in stop-over motion (Case 2).*

OpenRob enables a user to customize his own trajectory-planning algorithms and build them into the system for simulation and real-time control.

Controller Module

Six types of control algorithms have been built into the platform for computer simulation study and real-time implementation. They are the classical PID control, variant PID control, sliding-mode control, computed torque control, adaptive neural network control, and adaptive structural network control [10-14], as given in Table 1.

For the ease of software development and portability, the control algorithms are developed in DLL style. In each DLL, four main standard C style functions, `RunController()`, `Initialize()`, `Rollback()`, and `Tuning()` are declared as export functions. The features of these exported functions are described as follows.

- `Initialize()`: Initializes the parameters related to the controller.
- `Tuning()`: Invokes the friendly GUI for the user to tune the corresponding controller parameter.
- `RunController()`: Given the desired trajectory and the states, this executes the control algorithm to compute the output torque.
- `Rollback()`: De-initializes the controller-related parameters.

To ease the tuning of the controller parameters, GUIs as shown in Figures 4-10 have been designed. The user can easily change the gains by keying the values in the corresponding boxes.

As an open-architecture platform, OpenRob enables a user to customize his own control algorithm and implement it in the system for simulation and real-time control purposes. By selecting the "User Defined Control" option of the `Controller` module in `System Configurator`, a dialog box will be popped out for the user to select his desired control algorithm DLL file.

Case Studies

In order to demonstrate the functionalities of OpenRob and have a clear picture of the overall performance, several case studies are conducted in this section. In the first case, the kinematics and dynamics properties of two different robots are considered. The second case involves the `Trajectory Planner`, in which different trajectory-planning schemes are studied. In the third case, the control performances of different algorithms are presented.

Case 1: Study of Different Robot Models

In this case study, Planar (RR) and Cylindrical (RPP) robots are selected for preliminary study of ro-

Table 2. Controller Parameters Setting in Case 3					
Control Type	**Adjustable Gains and Parameters**				
Classical PID	K_D = diag[10]	K_P = diag[1000]	$K_I = 0$		
Variant PID	K_P = diag[10]	$K_I = 0$	K_1 = diag[100]	$K_2 = 0$	$K_3 = 0$
Computed Torque	K_D = diag[10] $\hat{P}_0 = 2$	K_P = diag[100] $\hat{P}_1 = 0$	$K_I = 0$ $\hat{P}_2 = 1$	$\hat{P}_3 = 15$	$\hat{P}_4 = 0$
Sliding Mode	K_r = diag[200,100]	K_1 = diag[100]	$K_2 = 0$	$K_3 = 0$	
Adaptive NN	K_P = diag[10] Γ_D = diag[0.001] $\text{Node}_D = 100$ $\sigma^2 = 10$	$K_I = 0$ Γ_C = diag[0.001] $\text{Node}_C = 100$ $\tau_r = 0$	K_1 = diag[100] Γ_G = diag[0.008] $\text{Node}_G = 100$	$K_2 = 0$	$K_3 = 0$
Adaptive SN	K_P = diag[10] Γ_i = diag[0.007]	$K_I = 0$ Γ_0 = diag[0.05]	K_1 = diag[100]	$K_2 = 0$	$K_3 = 0$

bot kinematics and dynamics. After selecting Planar (RR) or Cylindrical (RPP) from the robot type combo list box, and by single clicking on the `Robot` functional box, a robot property sheet with detailed information about the kinematic and dynamic parameters is popped out as shown in Fig. 11 and Fig. 12. This information is actually divided into six categories: `Link Parameters`, `Inertia Parameters`, `Limit Parameters`, `Kinematics`, `Jacobian`, and `Dynamics` parameters. Furthermore, in the `Kinematics` property page, one can enter the joint positions for each joint by clicking the corresponding button, and the position and orientation of the end-effector can be obtained directly from the property page; and in the `Jacobian` property page, by keying in position and velocity for each joint, the linear and angular velocities of the end-effector can also be updated automatically. It can be seen that, with the friendly GUI, the basic concepts of kinematics and dynamics can be observed and studied interactively.

Case 2: Investigation of Trajectory-Planning Schemes

In this case study, different trajectory-planning schemes are demonstrated. For simplicity, the planar robot with the configuration in Fig. 2 is used. Assuming the home position of the planar robot is $q_1 = 0$ and $q_2 = 0$, a series of via points has been specified with $q_1 = \{0, 50, -50, 0\}$deg and $q_2 = \{0, -100, 50, 0\}$deg. By selecting the planning category as joint-space planning and the motion type as stop-over or continuous motion, the generated trajectories by the `Trajectory Generator` with different profile schemes are shown in Figs. 13-18. The first column of the performance graph panes shows the desired joint position for each joint, while the second and third ones are the corresponding desired velocity and acceleration profiles.

It can be observed that although the position trajectories are almost the same in different schemes, the velocity and acceleration trajectories are totally different. In stop-over motion, all the joints start to move with zero velocities at the same time and reach the desired point simultaneously with zero velocities, then the motion continues in the next section and so on. In continuous pass-through motion, all the joints start from the initial point and go through each via point simultaneously with zero velocity and eventually stop at the final point with zero velocity.

Case 3: Testing of Different Control Algorithms

In this section, the performances of different control schemes are illustrated, such as classical PID control, variant PID control, sliding-mode control, computed torque control, neural-network-based adaptive control, and structural network-based adaptive control. For simplicity, a planar two-link

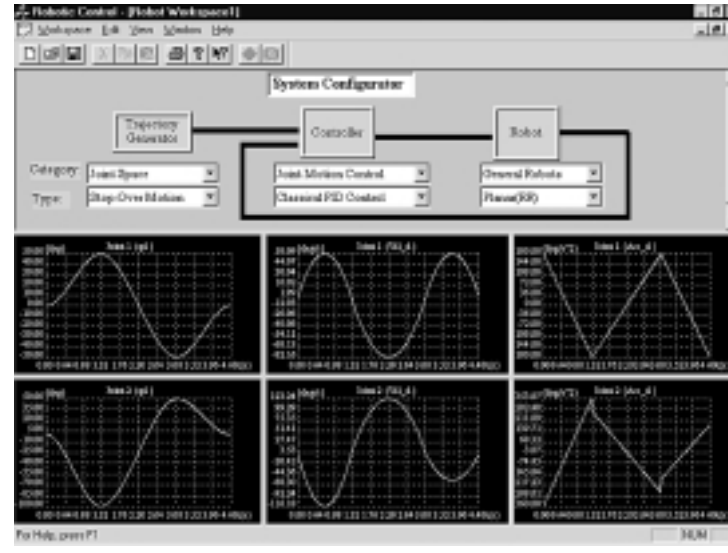

Figure 16. *Trajectory planned with the cubic method in stop-over motion (Case 2).*

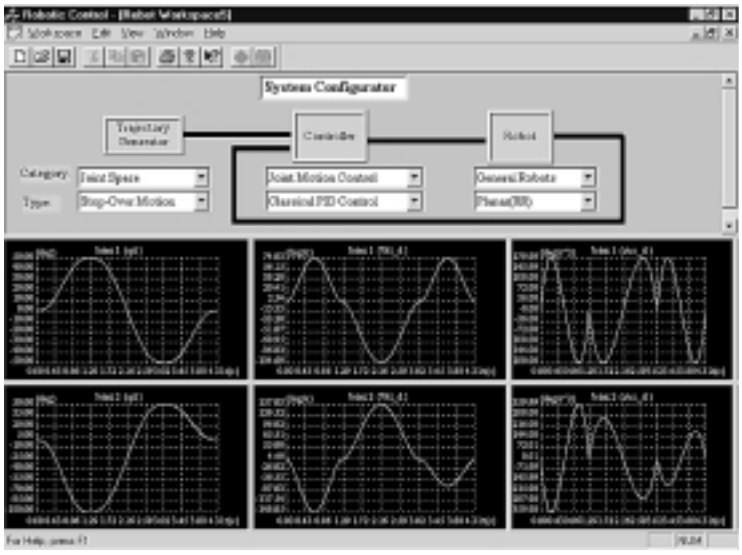

Figure 17. *Trajectory planned with the quintic method in stop-over motion (Case 2).*

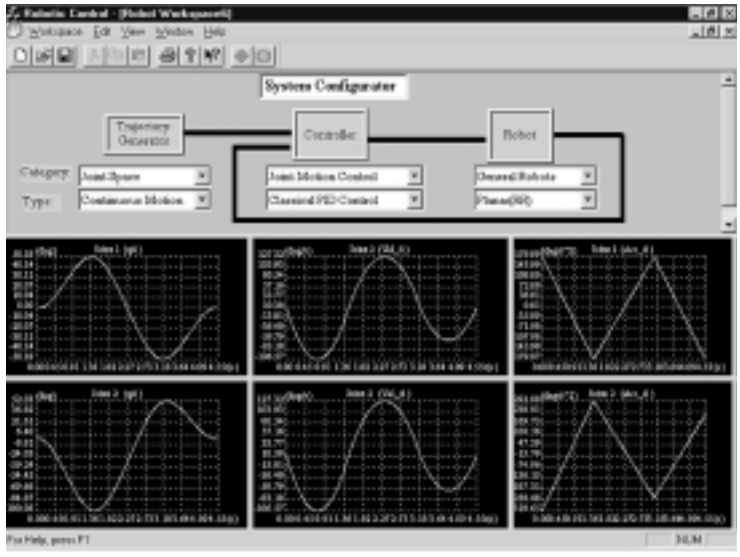

Figure 18. *Trajectory planned in continuous motion (Case 2).*

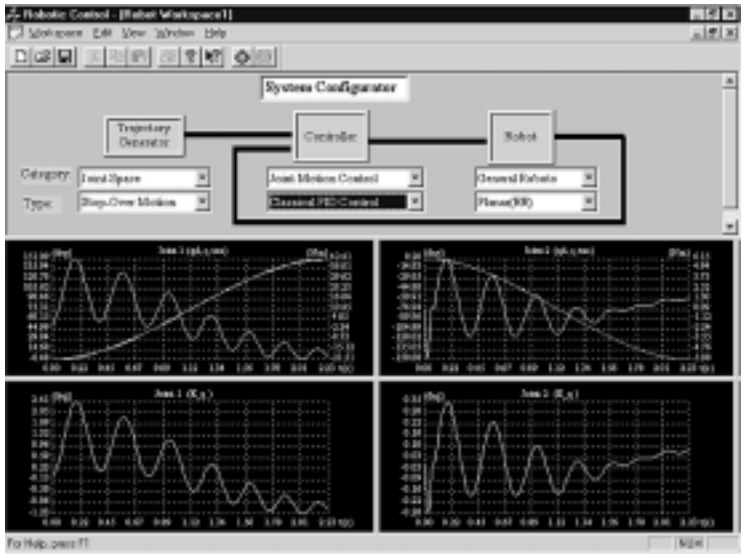

Figure 19. *Performance with classical PID control (Case 3).*

robot is chosen. The desired trajectory is generated using a cubic joint-space planning scheme by specifying the initial and final positions for the two joints as $q_o = [0\ 0]^T$ deg, $q_f = [150\ -150]^T$ deg. The control gains and adaptation parameters used for each control scheme are given in Table 2. The performances are shown in Figures 19-23. The performance graph pane is divided into four sections; i.e., (1) top-left corner shows the desired trajectory *qd*, the actual joint position response q, and the corresponding control signal τ for joint 1; (2) the bottom-left corner shows the joint tracking error for joint 1; (3) the top-right corner shows the desired and actual joint position response and the corresponding control signal for joint 2; and (4) the bottom-right corner shows the joint position error for joint 2. Note that the curves of different colors representing *q* and *qd* are very close to each other and appear to be only one line in a black and white printout such as here.

Summary

In this article we have presented OpenRob, an integrated open-architecture platform for computer-aided control system design of robotic control systems. The implementation issues have been described in detail. It provides a one-stop solution in the sense that model building, controller design, and simulation are integrated into one platform for the ease of real-time implementation. The open-architecture feature enables users to further develop custom robot models, trajectory-planning schemes, and control algorithms into the system for simulation study and real-time control. The software can also be used as a tool for educational purposes. Several case studies have been provided to demonstrate the features of this open-architecture system.

Keywords

Robotics, robot control, computer simulation, computer-aided control system design.

References

[1] *Workspace 4 Educational User Guide Manual*, Robot Simulations Ltd., UK, 1997.

[2] J. Nethery and M.W. Spong, "Robotica: A mathematica package for robot analysis," *IEEE Robotics and Automation Mag.,* vol. 1, no. 1, pp. 13-20, 1994.

[3] J. Nethery and M.W. Spong, "A mathlink-based front end for the Robotica package," *Mathematica J.*, , vol. 5, no. 2, pp. 72-79, 1995.

[4] J. Denavit and R.S. Hartenberg, "A kinematic notation for lower-pair mechanism based on matrices," *Trans. ASME J. Applied Mechanics*, vol. 77, pp. 215-221, 1955.

[5] M.W. Spong and M. Vidyasagar, *Robot Dynamics and Control*. New York: Wiley, 1989.

[6] K.S. Fu, R.C. Gonzalez, and C.S.G. Lee, *Robotics: Control, Sensing, Vision and Intelligence*. New York: McGraw-Hill, 1987.

[7] C.S. Lin, P.R. Chang, and J.Y.S. Luh, "Formulation and optimization of cubic polynomial joint trajectories for industrial robots," *IEEE Trans. Automatic Control*, vol. AC-28, no. 12, pp. 1066-1073, 1983.

[8] D.M. Himmelbian, *Applied Nonlinear Programming*, New York: McGraw-Hill, 1972.

[9] S.S. Ge and D.L. Gu, "Implementation of trajectory planner in OpenRob system," in *Proc. Int. Conf. Control, Automation, Robotics and Vision,* Singapore, Dec. 1998, vol. 1, pp. 524-528.

[10] J.J. Craig, *Introduction to Robotics: Mechanics and Control*. Reading, MA: Addison-Wesley, 1989.

[11] J.-J.E. Slotine and W. Li, *Applied Nonlinear Control*. Englewood Cliffs, NJ: Prentice Hall, 1991.

[12] F.L. Lewis, C.T. Abdallah, and D.M. Dawson, *Control of Robot Manipulators*. New York: MacMillan, 1993.

[13] S.S. Ge, T.H. Lee, and C.J. Harris, *Adaptive Neural Network Control of Robot Manipulators*, River Edge, NJ: World Scientific, 1998.

[14] R.P. Paul, *Robot Manipulator: Mathematics, Programming and Control*, Cambridge, MA: MIT Press, 1981.

S.S. Ge received the B.Sc. degree in control engineering from the Beijing University of Aeronautics and Astronautics, China, in July 1986, and the Ph.D. degree and the Diploma of Imperial College (DIC) in mechanical/electrical engineering from the Imperial College of Science, Technology, and Medicine, University of London, UK, in January 1993. From May 1992 to June 1993, he was a postdoctoral research associate at Leicester University, UK. He has been with the Department of Electrical Engineering at the National University of Singapore as a lecturer from July 1993 to June 1998 and as a senior lecturer since July 1998. He was a visiting staff member in the Laboratoire d'Automatique de Grenoble, France, in 1996 and the Department of Electrical and Electronics Engineering, the University of Melbourne, Australia, in 1998 and 1999. He served as an associate editor on the Conference Editorial Board of the IEEE Control Systems Society in 1998 and 1999 and has been serving as an associate editor of *IEEE Transactions on Control Systems Technology* since June 1999. He has authored and coauthored over 100 international journal and conference papers and one monograph, and he has coinvented three patents. He was the winner of the 1999 National Technology Award of the National Science and Technology Board. He serves as a technical consultant to industry. His current research interests are adaptive control, neural networks and fuzzy logic, robot control, real-time implementation, genetic algorithms, friction compensation, and sensor fusion.

T.H. Lee received the B.A. degree with First Class Honours in the Engineering Tripos from Cambridge University, UK, in 1980, and the Ph.D. degree from Yale University in 1987. He is the head of the Con-

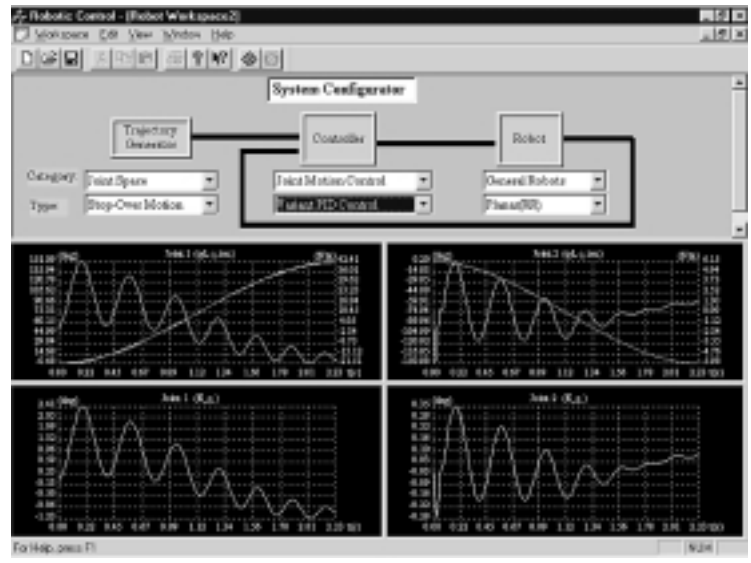

Figure 20. *Performance with variant PID control (Case 3).*

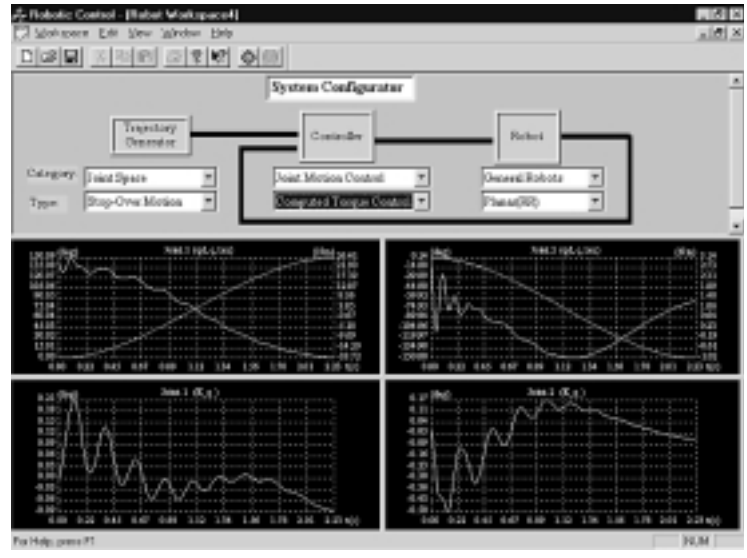

Figure 21. *Performance with computed torque control (Case 3).*

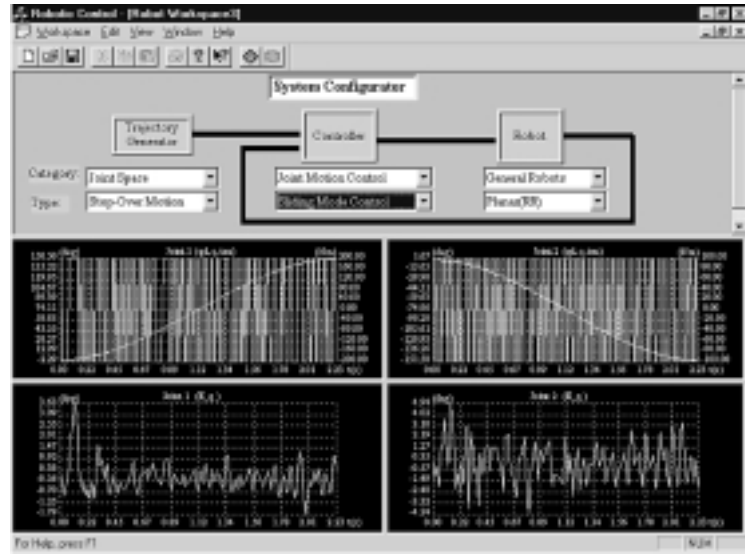

Figure 22. *Performance with sliding-mode control (Case 3).*

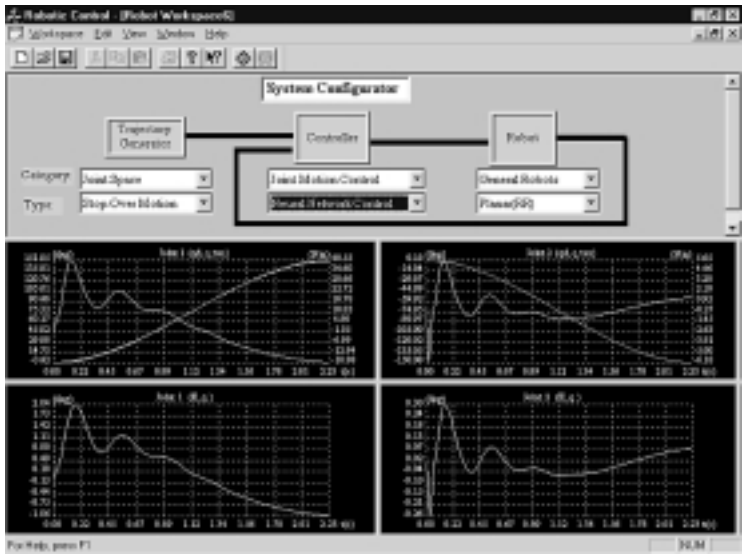

Figure 23. *Performance with neural-network-based adaptive control (Case 3).*

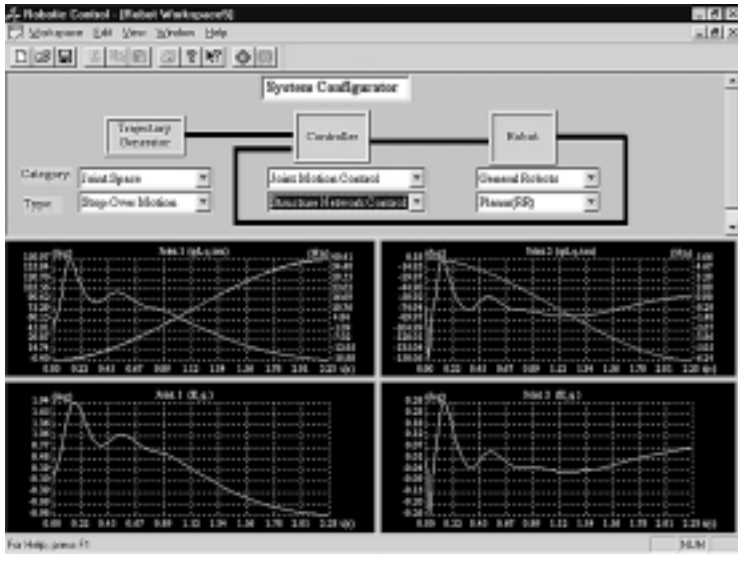

Figure 24. *Performance with structural-network-based adaptive control (Case 3).*

trol Division of the Department of Electrical Engineering of the National University of Singapore, and he is also the vice-dean (research) in the Faculty of Engineering there. Dr. Lee's research interests are in the areas of adaptive systems, knowledge-based control, and intelligent mechatronics. He has published over 100 technical papers in international journals and conference proceedings in these areas, and he currently holds associate editor appointments in *Automatica* (an IFAC Journal); the *IEEE Transactions on Systems, Man and Cybernetics*; *Control Engineering Practice* (an IFAC journal); and the *International Journal of Systems Science*. He is also the regional editor (for the Far East) of *Mechatronics* (Oxford, Pergamon Press). Dr. Lee was a recipient of the Cambridge University Charles Baker Prize in Engineering.

D.L. Gu was born in Zheng Jiang, Jiangsu Province, P.R. China, in 1969. He received his B.Eng. degree in scientific instrument engineering from Zhejiang University, P.R. China, in 1991. He is currently an M.Eng. student in the Department of Electrical Engineering, the National University of Singapore, and with Hewlett-Packard Singapore Pte. Ltd. as a research engineer. His research interests include robotic control, neural network control, and control applications.

L.C. Woon received the B.Eng. degree in electrical engineering with First Class Honours in 1996 and the M.Eng. in 1998, both from the National University of Singapore. His main research interests are in the areas of control of robot manipulators, neural network control, and real-time control systems.

Address for Correspondence: S.S. Ge, Department of Electrical and Computer Engineering, National University of Singapore, Singapore 119260. E-mail: elegesz@nus.edu.sg.