PERGAMON

Robotics and Computer Integrated Manufacturing 17 (2001) 355–365

Robotics and Computer Integrated Manufacturing

www.elsevier.com/locate/rcim

A PC-based open robot control system: PC-ORC☆

Keum-Shik Hong[a,*], Kyung-Hyun Choi[b], Jeom-Goo Kim[c], Suk Lee[a]

[a] School of Mechanical Engineering, Pusan National University, 30 Changjeon-dong, Kumjeong-ku, Pusan, 609-735, South Korea
[b] School of Mechanical Engineering, Cheju National University, Cheju, 690-756, South Korea
[c] Department of Mechanical and Intelligent Systems Engineering, Pusan National University, Pusan, 609-735, South Korea

Received 1 October 2000; received in revised form 15 March 2001; accepted 30 March 2001

Abstract

An open architecture manufacturing system pursues to integrate manufacturing components on a single platform. Therefore, a particular component can be easily added and/or replaced. In this paper, a modular and object-oriented approach for the PC-based open robot control (PC-ORC) system is investigated. A standard reference model for controlling robots, which consists of a hardware platform, an operating system module, and various application software modules, is first proposed. Then, PC-ORC system, which can reconfigure the control system in various production environments, is developed. The PC-ORC is constructed based upon the object-oriented method. Hence, it allows an easy implementation and modification of various modules. The PC-ORC consists of basic software, application objects, and additional hardware devices on a PC platform. Finally, by applying the proposed PC-ORC to a SCARA robot, the performance of the PC-ORC is examined.

Keywords: PC-based controller; Object-oriented method; Open architecture control; Robotics; Flexible manufacturing

1. Introduction

A modern manufacturing system should have the capability to meet consumer's various demands to new products within a certain designated time. Also, it should be able to accommodate the rapid development of computer technologies. Therefore, it pursues the open modular structure, which permits the flexibility, the integrability, and the concurrency of diverse manufacturing elements [1–9]. Under the prevalence of various manufacturing equipment and software resources, an open manufacturing system aims at an easy integration of multi-vendor equipment under a single standard platform.

Even though most current automation equipment such as robots and NC machine tools used in the manufacturing industries are programmable and may have various functions related to a given task, they are still characterized by their own control schemes, i.e., vendor-oriented control architectures. Therefore, when a number of machines from different vendors are combined, not only the lack of flexibility due to the incompatibility problem but also the waste of resources and the inefficiency due to duplicated peripherals are apparent.

In order to deal with the incompatibility problem arising from various vendors, a number of organizations have proposed the establishment of standards for the control systems used in the automation area. Also, it was acknowledged that too specific standards might restrict the willful initiations from vendors for new technologies. But, due to the reason that some existing hardware and software may have to be abandoned, such proposals didn't get much attention throughout the manufacturing society.

As an alternative approach, researchers have focused on developing an open architecture control system, which requires its control environment to conform to an open technology. As far as integration and interoperability are concerned, this approach does not force all manufacturing vendors to conform to one standard. Instead, this approach uses existing resources and realizes user-specific functions. One of the successful results along this approach is the Open Modular Architecture Controller (OMAC) instigated by GM, Ford, etc. The OMAC specifies the requirements for the manufacturing equipment used in the automotive

☆This work was supported in part by the Korea Research Foundation under Grant KRF-2000-EA0096. The third author was supported by the Brain Korea 21 Program.

*Corresponding author. Tel.: +82-51-510-2454; fax: +82-51-514-0685.
E-mail address: kshong@hyowon.pusan.ac.kr (K.-S. Hong).

0736-5845/01/$ - see front matter
PII: S0736-5845(01)00010-2

industry [7]. In the machine tool industry, the Open Systems Environment for Control (OSEC) has been suggested as an open architecture reference model for open architecture NCs and FA systems [8]. Also, European countries proposed the Open System Architecture for Controls within Automation systems (OSACA) as a reference platform for FA systems [9].

While the main stream in the research area of the open architecture system was on NC machine tools, the research for open robot controllers is getting more attention recently, because robots are used in many applications and are directly related to the production efficiency. Also, the virtual simulation environment for controlling as well as for configuring robot systems enables now the rapid increase of production efficiency [10,11].

As illustrated in Fig. 1, the structure of an industrial robot system, in general, includes: task control, manipulator, manipulator sensor interface, memory, sequencer, servo controller, external sensor interface, computational unit, auxiliary device interface, path generator, and power unit [13]. These components can be categorized into two major modules, the management module and the servo module, based upon the necessity of real time operation. The management module does not require the real time operation, because it manages the robot from the functional point of view by simply executing user's commands. On the other hand, the servo module concerns the control of the positions and speeds of robot arms, thus it requires the real time operation.

This paper presents a PC-ORC system, which is based upon the OSACA reference model. The proposed system incorporates a commercial simulation environment into its architecture for easier programming and verification. To enhance the modularity and reusability, the object-oriented technique is utilized in constructing the PC-ORC. Finally, in order to demonstrate the applicability of the PC-ORC, the proposed configuration is applied to an industrial SCARA robot.

This paper is structured as follows: In Section 2, a new reference platform for the PC-ORC is proposed. The characteristics of component modules are described. In Section 3, by analyzing the structure of an industrial robot, implementation strategies for the PC-ORC are described. In Section 4, a new approach for the robot motion program and a new conceptual user interface scheme are described. In Section 5, by applying the proposed PC-ORC to a SCARA robot, the applicability and the performance of the PC-ORC are examined. Finally, Section 6 describes the concluding remarks.

2. A Reference model for the PC-ORC

As depicted in Fig. 2, a new reference model for the PC-ORC, based upon the OSACA reference model [9], is proposed. The overall structure in Fig. 2 is basically identical to the OSACA system. However, because industrial robots vary from vendor to vendor, the management module configures not only hardware specifications but also retrieves necessary information regarding a particular robot system from the database. This new structure focuses on providing a one-stop solution for model building, controller design, and numerical simulations for the ease of real-time implementation. Three main parts of this model are a hardware platform, an operating system module, and application software modules. The role of these three parts is briefly explained.

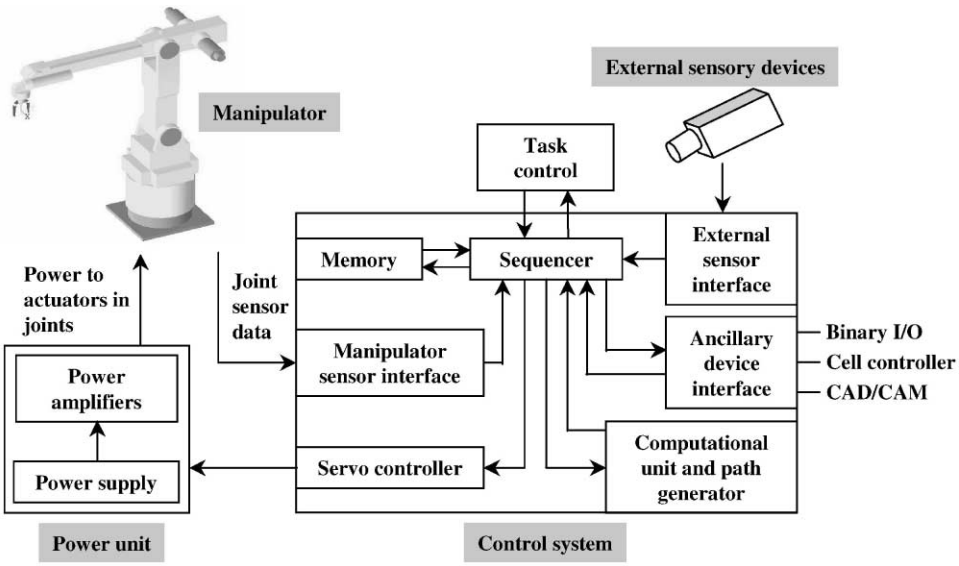

Fig. 1. Various components of a typical industrial robot system.

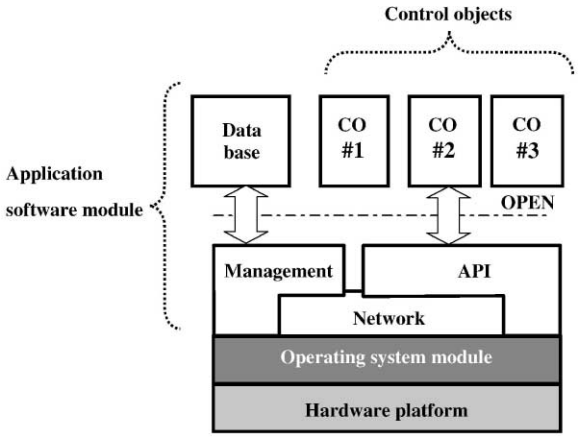

Fig. 2. A proposed standard reference model for the PC-ORC.

2.1. Hardware platform

A typical hardware platform consists of one (or more) processor(s), I/O units for the communication with external devices, data memory, and peripheral devices for user interface. In order to achieve the openness of the hardware platform, its components should comply with standardized specifications. However, since hardware technologies are changing rapidly, the openness of the hardware platform should be accomplished in their drivers, not by the hardware itself, to ensure the low cost and easy use.

PCs have been used as a standard platform for open architecture control system. The use of PC is expected to increase rapidly. Main reasons for the wide spread PC-based controls are the emergence of control network standards for motion and the I/Os that allow easier integration of various peripheral devices. Also, the easy modification and extension of operation software is another important factor for the PC-based control [2,12].

2.2. Operating system module

The operating system module should provide the hardware platform with the accessibility to the hardware, the application software, and application programming interface (APIs) developed by other vendors. In order to utilize a single-processor PC for the open architecture control system, the operating system module should be a real time operating system (RTOS). Core differences between the RTOS and the Windows (CE, NT, etc.)/DOS are the management and allocation method of the CPU time, responding method to interrupt, and resource sharing method during the operation of mechanical devices.

2.3. Application software modules

In an open architecture control system, the functions of the application software module should be allowed to be extended or to be modified. They are also required to satisfy the object-oriented characteristics to exchange data among the functions. Due to these requirements, the application software module is decomposed into four sub-modules: management, API, network, and control objects.

(1) *Management*: One essential characteristics of the open architecture control system is the ability of reconfiguration. The management object deals with this reconfiguration problem of the open architecture controllers. The first step of the reconfiguration process is to analyze the functions and performances of the various system components, as shown in Fig. 2. Then, a configuration order is generated by setting new values for the variables in the open architecture reference model by using the configuration editor. An appropriate value of the variable for the specified control system is fetched from the database that contains information about the configuration of the various manufacturing devices. This reconfiguration function is essential to the open control system to ensure that the system is able to cope with the manufacturing environment changes.

(2) *API*: The manufacturing environment requires the reconfiguration of the existing manufacturing devices and the addition of new ones according to the change of products and production processes. To satisfy this requirement, API provides the common interface to the existing hardware or software so that the application programs can be easily modified. That is, API should allow the 'plug and play' capability.

(3) *Network*: The network performs as a data bus when reading the status of an object or assigning an appropriate value for some necessary movement. There are two entities using the network: The client delivers a command to perform an appropriate function and the server executes the command requested by the client [1].

(4) *Control object*: The control objects mean the components used in controlling the real manufacturing devices from the software point of view. By assigning new values to the attributes of these control objects, the entire control system can be constructed and modified as required for the new manufacturing environment. Also, the control object should be hardware independent.

3. Implementation of the PC-ORC

This section presents the strategy to implement the standard reference model for the PC-ORC. More specifically, implementation strategies for the operating system module and the application software module are given by examining the structure of an industrial robot.

Also, the integration method of control objects is described.

3.1. Implementation of the operating system module

As an operating system of the PC-ORC, Windows NT is adopted because of its stability on PC platforms. However, Windows NT has some shortcomings in real-time operation [14], because it is difficult to guarantee the computation cycle of within several milliseconds for the servo control of periodic sensor sampling and control command calculation. In order to circumvent this problem, the motion controller of a servo board is dedicated for the servo system, while Windows NT handles the tasks that require relatively slow responses in robot motion planning and path generation.

In this paper, as the dedicated servo system, a motion controller of PMAC with a dual-port RAM is used [15]. The dual-port RAM acts as a buffer between the PMAC and Windows NT and allows downloading the robot path data from the PC memory into the buffer memory of PMAC at high speed [15]. During the servo system operation, the controlled values of each joint and the overall status of the servo system are fed back to Windows NT. The buffer memory is capable of storing 2000 joint values, and its size is adjustable if needed.

3.2. Implementation of the application software module

As a methodology for developing application software modules, an object-oriented modeling paradigm is employed, which defines individual module components as objects and assigns appropriate attributes to the defined objects [16]. As shown in Fig. 3, classes of the defined objects and the relationship among the classes are represented by using the object-oriented modeling techniques (OMTs). In Fig. 3, '◇' denotes the aggregation relation and '—' means the logical link [17,20].

The Sequencer object converts the commands received from the ORIS editor (to be described in Section 4) to the format that can be utilized by the related objects. This task is completed with the help from the TaskControlConverter object and the MotionControlConverter object. The TaskControlConverter object transforms the task information transmitted by the Sequencer object into the commands that can be recognized and executed by the TaskControl object. Meanwhile, the MotionControlConverter object generates a file to be downloaded into the PMAC buffer. The TaskControl object interacts with the Point object, the PathGeneration object, and the InverseKinematics object in order to generate the simulation information for the Simulation object. In addition, it receives an executable robot trajectory as a result of the trajectory modification via the 3D graphic simulation. The Point object stores the position and orientation of a body in the 3D space with respect to the world coordinate. Also, this object is responsible for teaching a virtual robot for program verification. The PathGeneration object generates the trajectory of the robot end-effector with a series of stored position/direction information. In case that the tool position is expressed in the world

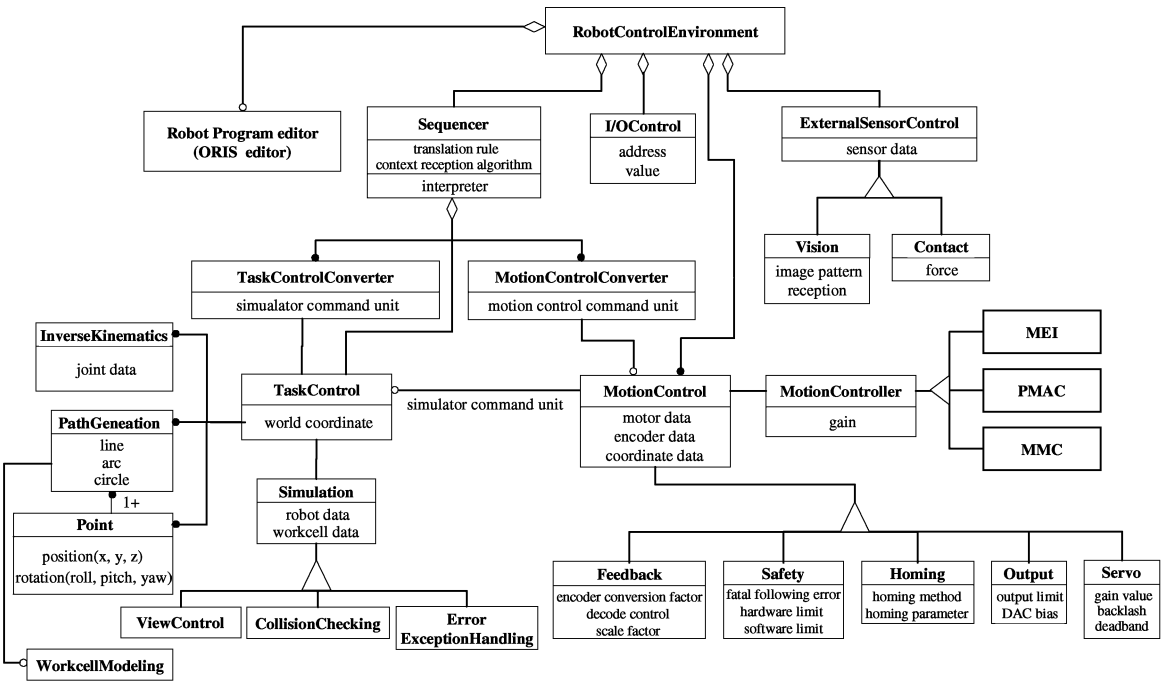

Fig. 3. Class structure of the application software module.

coordinate, the InverseKinematics object transforms the tool position into the joint coordinate.

The Simulation object provides a virtual simulation environment where the intended robot tasks can be examined before applying to the real robot. To this end, this object has functions to model robots and work cells, to check for a possible collision, and to handle error conditions. The Simulation object consists of two sub-objects (not shown in Fig. 3); the CIMulation made in CODE [18] and the interface dialog box developed in C++. The CODE is commercial software for robot simulation and its API provides the interaction between these two objects.

The MotionControl object deals with the environment configuration task of the actuators installed in a robot system as well as the verification of the actuator status. This object interacts with various motion controllers such as PMAC, MEI, and MMC via the MotionControl object that is responsible for hardware configuration such as adjusting motor outputs and gains. Also, the MotionControl object interacts with the Feedback, Safety, Homing, Output, and Servo objects whose functions are illustrated in Fig. 3.

The I/O control object receives or sends signals from/to the peripheral equipment (conveyor, NC machine, and cell controller) through input or output ports, respectively. The ExternalSensorControl object accepts external status of a robot system from the external sensors and converts them into the internal format.

3.3. Integration of application software modules

The PC-ORC has been developed through the integration of application objects, the CODE system, and hardware including PMAC, MEI, and MMC after completing the PC hardware platform and the operation system module. Also, the TCP/IP protocol socket provided in C++ is used to read and write the data among objects of the application software module. This approach employing the TCP/IP sockets aims to guarantee the open architecture for adding and modifying objects. Even though the TCP/IP protocol is functionally limited compared with the Manufacturing Automation Protocol/Manufacturing Message Specification (MAP/MMS), it is widely accepted in many areas because it is inexpensive and technically very mature. Moreover, in case of a distributed environment where objects are handled in different computers, the combination of Ethernet and TCP/IP is readily available for communication among objects only with a server program. Fig. 4 shows such a server program, in which each application object (i.e., client) is represented in a dialog box. As shown in Fig. 5, the PC-ORC includes a configuration editor, which configures the attributes of each object with TCP/IP. Fig. 6 shows the overall structure of the PC-ORC.

4. Robot programming for the PC-ORC

4.1. Open robot instruction set

Current approaches to robot motion programming are classified into two major methods [19]. One approach is the teach-and-playback method, which uses an on-line teach pendant. The other approach is to use an off-line high level programming language. The advantages of the teach-and-playback method are that it requires only a relatively small memory space to record positions and that it is simple to learn and suitable for simple tasks. However, the main disadvantage is that it is difficult to integrate sensory feedback information into the robot program. In addition, if the task is changed, the whole manufacturing system should be interrupted during the robot programming. On the other hand, the off-line approach can deal with the programs for complicated tasks, while it requires operator's expertise and experience in dealing with the robot language.

In order to overcome these problems, the Open Robot Instruction Set (ORIS) is proposed in this paper. The ORIS is based on a format similar to the syntax of English language, which allows the instruction to be easily understood. As shown in Table 1, there are four command units for the ORIS.

(1) *Move control command unit*: This command unit is used to describe the motion of a robot end-effector. The commands are classified into four categories according to four types of basic motions: linear interpolation motion, joint interpolation motion, continuous path motion, and gripper motion.

(2) *Configuration control command unit*: The configuration commands are used to define its configuration such as maximum joint speed and acceleration and various control modes.

(3) *Logic device control command unit*: A robot must be able to communicate with other auxiliary devices. The logic device control commands provide means for setting, resetting, and toggling through a specific communication line that is connected with sensors and actuators. In addition, the commands for operating a vision system are included in this command unit.

(4) *Program control command unit*: The program control commands are necessary in dealing with problems associated with an unexpected event and an exceptional condition during the robot operation. These commands provide means to control the program flow like conditional branches in a high-level computer language.

The ORIS has been implemented in the object-oriented programming (OOP) language C++. Each object contains data structures and operations that express the behavior of the object. All the entities

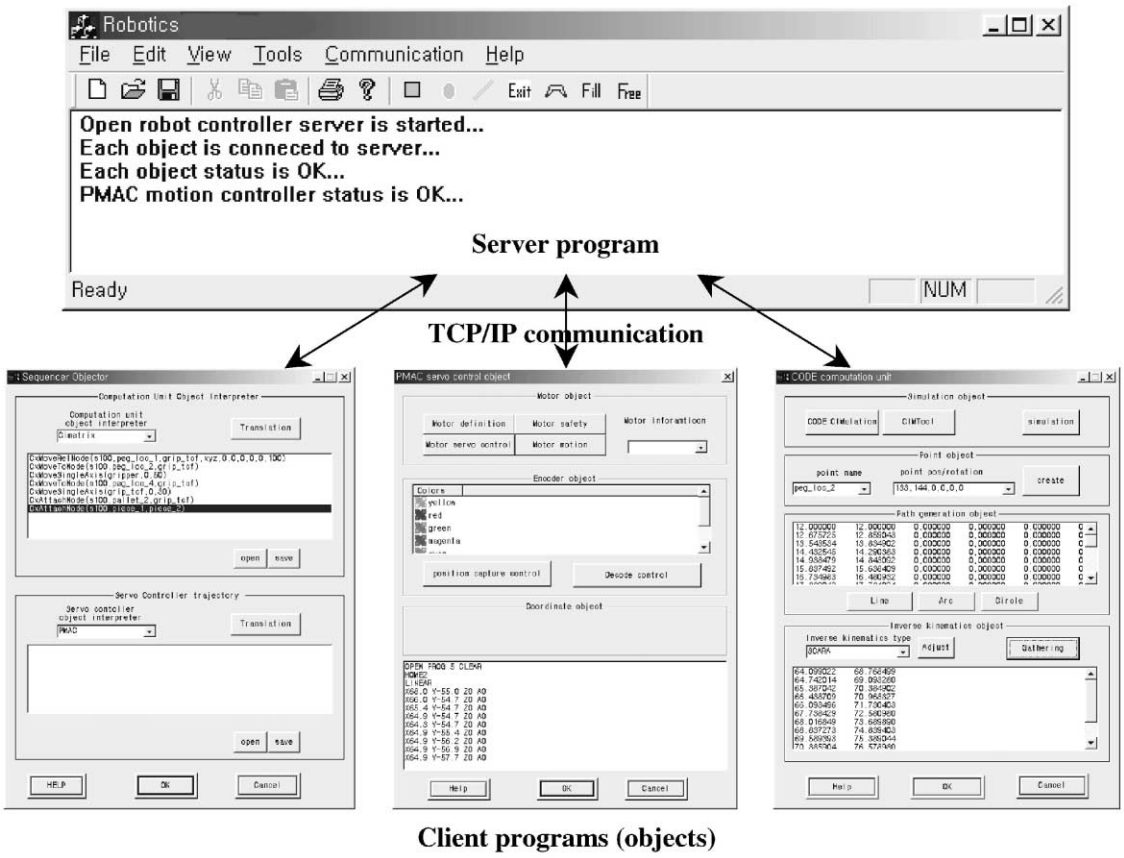

Fig. 4. Client-server characterization of the application software module.

Fig. 5. Configuration editor of the PC-ORC.

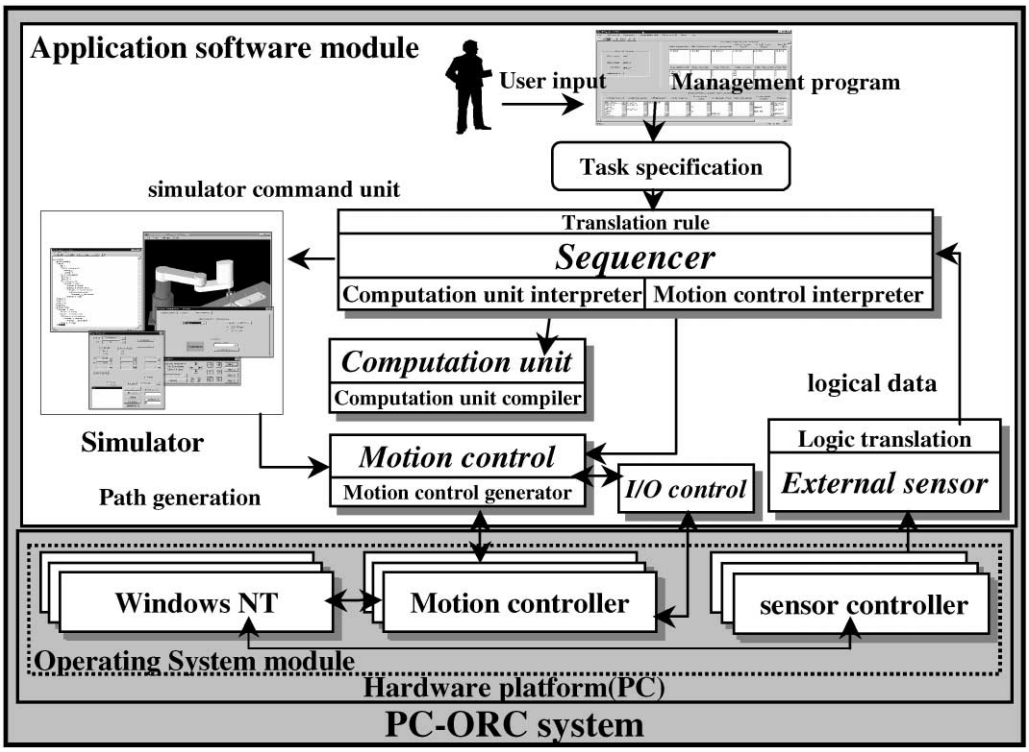

Fig. 6. Components of the PC-ORC.

involved in the robot system are represented as objects. The OOP features of polymorphism, inheritance, abstraction and data encapsulation were instrumental in the development of the ORIS, in such a way that new commands can be easily added without affecting other commands.

Fig. 7 shows the ORIS Editor where a robot motion program can be developed using the ORIS commands. The List Box (three boxes on the top in Fig. 7) shows various commands in a hierarchical structure. The Edit Box (the box in the second row of Fig. 7) is used for the creation of a robot motion program, displaying the sequence of a robot motion. The program can be written by selecting an appropriate command in the List Box, which, in turn, opens child windows for required parameters. Furthermore, by using the line command input line box, the user who is familiar with the ORIS can directly type the commands. This type of interactive input via GUI allows users to conveniently construct robot programs.

4.2. Conversion of robot motion program

The robot motion program written in the ORIS format needs a conversion into the appropriate formats for related objects. The robot motion program is classified according to the command unit type and stored into the proper fields of the database. By using predefined conversion rules, these motion programs are transformed into the sequence of commands expressed in terms of related library, CODE API, and PMAC. Then, these transformed commands are used to directly drive the control objects. Fig. 8 shows an example of the conversion process. This conversion process is conducted by the TaskControlConverter and MotionControlConverter objects.

5. PC-ORC application to a SCARA robot

To order to demonstrate the applicability of the PC-ORC, the developed system has been applied to a SCARA robot. The performance of the PC-ORC can be evaluated from two different aspects. The first aspect is to examine how easy system modifications are. To this end, a vision system is added to the existing robot control system developed with the PC-ORC. The second aspect is to examine the performance of the real time control.

5.1. Hardware construction

Fig. 9 depicts the hardware structure of the PC-ORC, which includes a hardware platform PC, a motion

Table 1
ORIS commands

Commands		Command unit
Move control command unit		
Linear interpolated motion	MoveTo	MoveTo Tool(gripper_tcf) to TargetPoint(point_name)
	MoveNear	MoveNear Tool(gripper_tcf) relativeto TargetPoint(point_name) with Offset(x,y,x, roll, pitch, yaw)
	MovePath	MovePath Tool(gripper_tcf) according to Path(Path_name) with Offset(x, y, z, roll, pitch, yaw)
	MoveCircle	MoveCircle Tool(gripper_tcf) to TargetPoint(point_name)
CP motion	Continous path generation	CP_Generation(Source, File_Name)
Joint interpolated motion	MoveSingleAxis	MoveSingleAxis Joint(JointNum) with Const(value)
	MoveAllAxes	MoveAllAxes Joints(Joint_1, Joint_2, Joint_3 ...) with Const(value_1, value_2, value_3 ....)
Gripper motion	Open	Open Gripper(gripper_name) with Const(value)
	Close	Close Gripper(gripper_name) with Const(value)
Configuration control command unit		
Set		Set Mode(joint/linear)
		Set JointSpeed(value)
		Set TipSpeed(value)
		Set ScrewSpeed(value)
		Set ScrewAccel(accel_value, decel_value)
		Set JointAccel(joint_num, accel_rise, accel_fall)
Logic device control command unit		
Vision Operation Commands		Board_Initialize(address, value)
		Capture_Image(value)
		Prefilter(value)
		Detect_boundary(value)
		Erosion_boundary(value)
		Save_Data(File_name)
Send		Send Signal(signal_num, value)
Get		Get Signal(signal_num)
Wait		Wait Signal(signal_num, compare_value, timeout_value)
Program control command unit		
OnState		OnState(condition) ~ stop/continue
IfThen		IfThen(condition_compare) ~ (other command unit)
While		While(condition) ~ exit

controller PMAC, and a vision board. Fig. 10 shows the actual system components.

5.2. Software construction

(1) *Reconfiguration environment and motion program*: Attributes of control objects are reestablished to be suitable for the SCARA robot system using the Environment Configuration Editor, and the hardware and operation commands associated with the vision systems are added. These reconfiguration and system modification tasks are done by specifying the characteristics of the related hardware such as servo driver and vision system. Therefore, this reconfiguration allows the PC-ORC to be applied to various types of robot systems and manufacturing environment. To evaluate the performance of the integrated control system, an example task is executed as follows:

- The PC-ORC obtains image information from the vision system, downloads them into memory, and extracts the robot trajectory.
- Based on this trajectory, the robot path is generated by applying the appropriate coordinate transformation.
- The virtual simulation environment takes and modifies the given robot path, and generates the data file which is downloaded into the buffer of PMAC.
- The path information stored in the buffer is retrieved to drive the periodic servo-loop on PMAC.
- The execution results of the PMAC are fed back to PC.

Fig. 7. A dialog box of the motion program editor.

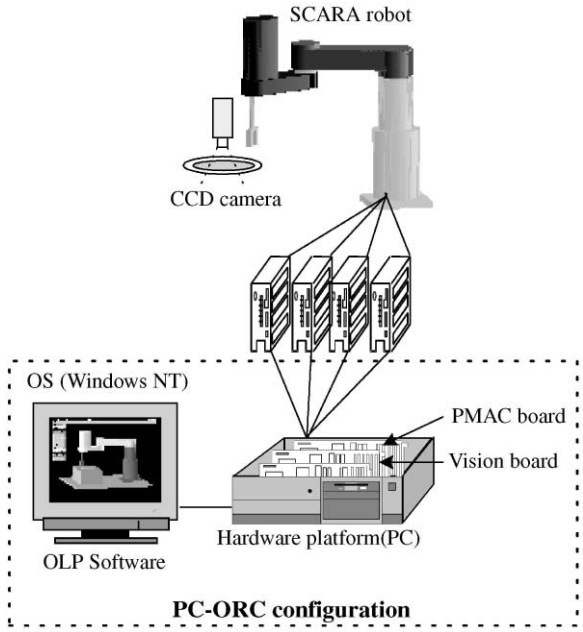

Fig. 9. Hardware configuration of the PC-ORC applied to a SCARA robot.

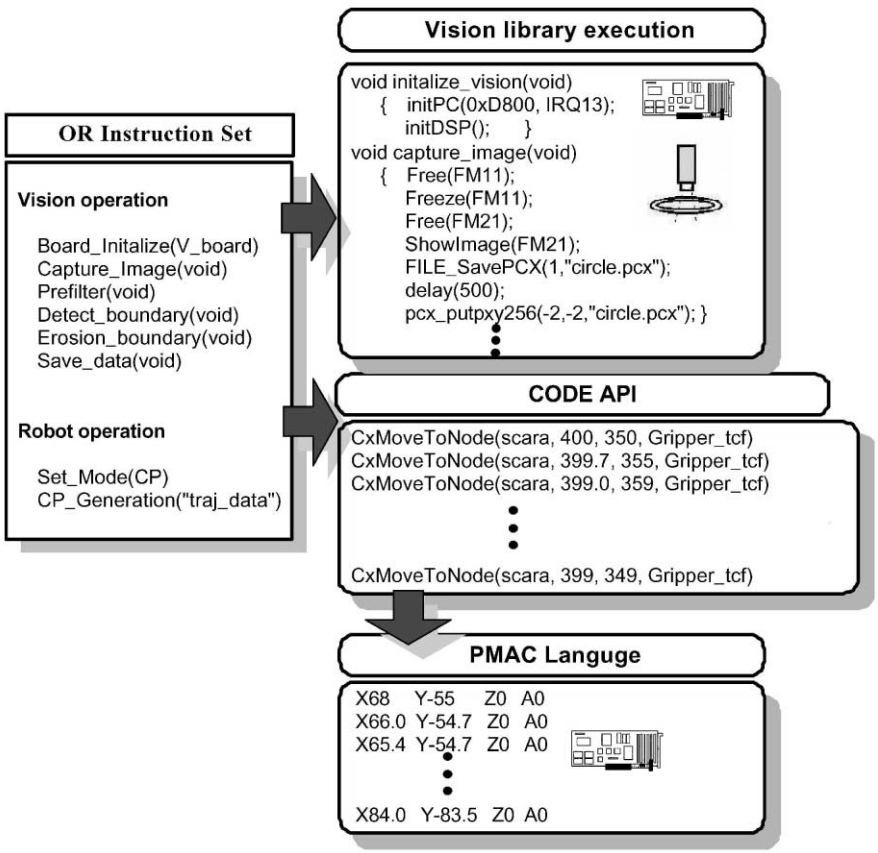

Fig. 8. An example of motion programming using the ORIS.

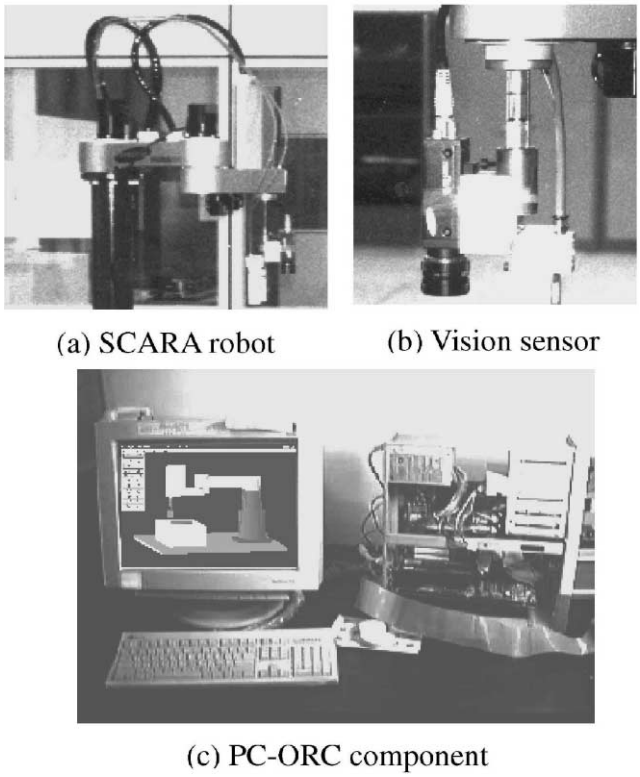

Fig. 10. Pictures of the PC-ORC with a SCARA robot.

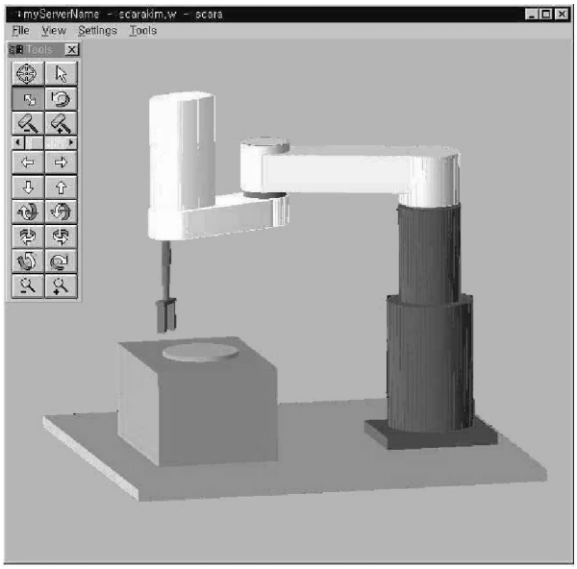

Fig. 11. A virtual SCARA robot configured with the CODE system.

(2) *Virtual simulation using the CODE*: A virtual robot and an operation environment created with 3D graphics are needed for off-line teaching, path planning, simulation, and performance evaluation. For this purpose, the PC-ORC provides a simulation environment that interacts with the CODE system. As shown in Fig. 11, a virtual SCARA robot and its operation environment

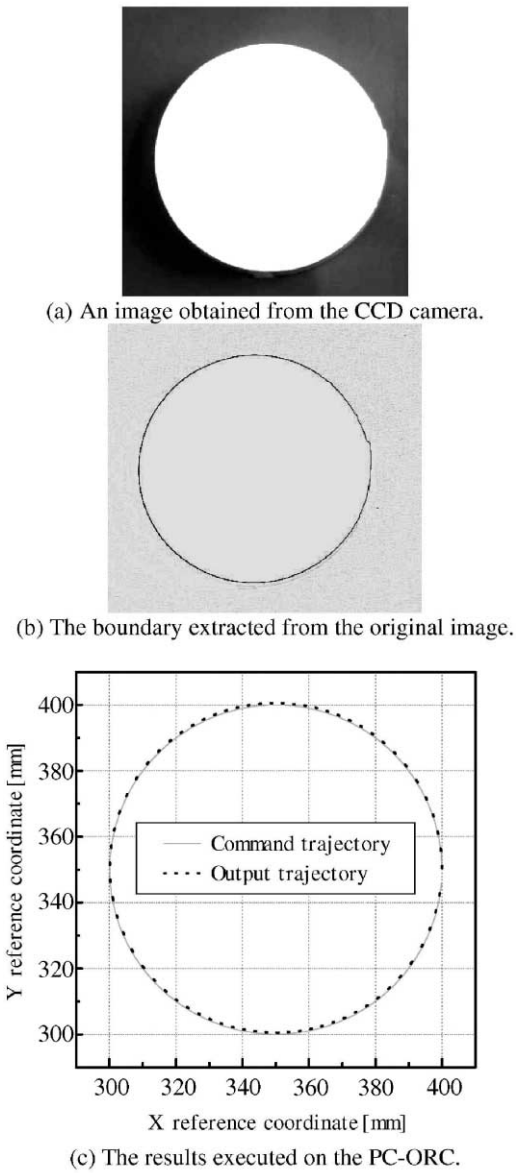

Fig. 12. Trajectory extraction process and the actual trajectory obtained with the PC-ORC.

are modeled using the CODE, and simulated according to the set of robot motions specified by the operator. During the simulation, the robot path is updated and consequently sent to the PMAC for motion control.

(3) *Performance evaluation*: The PC-ORC is able to replace the closed architecture control system of the SCARA robot with the PC-ORC. In addition, it is relatively easy to integrate a vision system. Fig. 12 shows the detailed performance of the PC-ORC and a vision system. Fig. 12(a) illustrates the image information obtained from the vision system, part (b) shows the trajectory to be followed, which is extracted from the image information, and part (c) depicts the actual robot path generated by the SCARA robot. Figs. 13 and 14 show response performances of the first and second axes

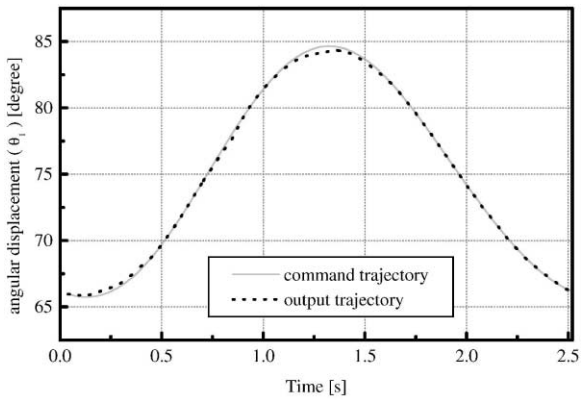

Fig. 13. Tracking performance of the first axis of the SCARA robot.

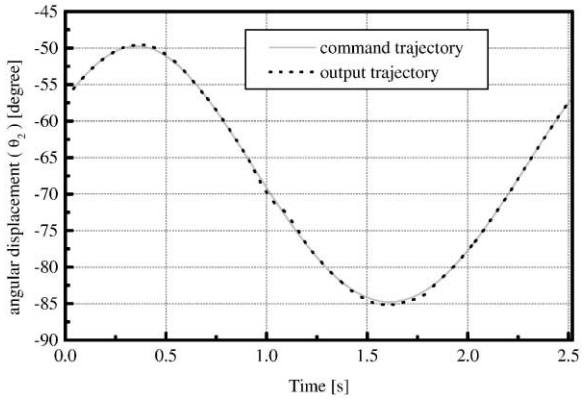

Fig. 14. Tracking performance of the second axis of the SCARA robot.

or joints, respectively. These results demonstrate that the PC-ORC can be applied to the real time control of an industrial robot. Also, it is found that the virtual simulation can lead to the enhanced efficiency of robot task planning and adaptation to the new manufacturing environment.

6. Conclusions

This paper describes an open control environment, the PC-ORC, which integrates the various control functions of industrial robot control systems into a single unified environment along with its evaluation. The PC-ORC is constructed by implementing object-oriented application software and commercial software on a standard PC platform. Since the PC-ORC is based on a PC and implements the software modules and objects by using object-oriented paradigm, it provides the development environment for rapidly building a new robot control system. In addition, the PC-ORC allows the easy integration and reuse of hardware and software. Because the motion and external sensor controller cards are installed and operated on the PC platform, however, the control system shows the limited stability and robustness compared with the closed architecture control system. In the near future, it is expected that an operating system with more stability and capability can make the open architecture control system applicable to the real industrial manufacturing environment.

References

[1] Sperling W, Lulz P. Enabling open control systems: An introduction to the OSACA system platform. ESPRIT III Project, FISW GmbH, Stuttgart, 1995.
[2] Wright PK. Principles of open-architecture manufacturing. J Manuf Syst 1995;14(3):187–202.
[3] Marchetti LA, Wright PK. A PC-based open architecture controller: Design, implementation, and operation. ASME 1999 International Mechanical Engineering Congress and Exposition 1999, vol. 10, p. 863–7.
[4] Fernandez JA, Gonzalez J. The NEXUS open system for integrating robotic software. Robotics Comput Integrated Manuf 1999;15(6):431–40.
[5] Pires JN, Sá da Costa JMG. Object-oriented and distributed approach for programming robotic manufacturing cells. Robotics Comput Integrated Manuf 2000;16(1):29–42.
[6] Mak KL, Lau HYK, Wong STW. Object-oriented specification of automated manufacturing systems. Robotics Comput Integrated Manuf 1999;15(4):297–312.
[7] OMAC API Work Group, OMAC API SET-Working Document, OMAC API Work Group, Ver. 0.18, 1998.
[8] OSEC, Open Systems Environment for Controller, http://www.sml.co.jp/OSEC.
[9] OSACA, Open System Architecture for Controls within Automation Systems, ESPRIT III Project 6379/9115, 1996.
[10] Burdea GC. Invited review: The synergy between virtual reality and robotics. IEEE Trans Robotics Autom 1999;15(3):400–10.
[11] Freund E, Rossmann J. Projective virtual reality: Bridging the gap between virtual reality and robotics. IEEE Trans Robotics Autom 1999;15(3):411–21.
[12] Babb M. PCs: The foundation of open architecture control system. Control Engineering 1996;75–82.
[13] Klafter RD, Chmielewski TA, Negin M. Robotic Engineering: An Integrated Approach. Englewood Cliffs, NJ: Prentice-Hall, 1989.
[14] Julian T. Beginning Windows NT Programming. Wrox Press Inc., 1998.
[15] PMAC, PMAC User's Manual, Delta Tau Inc., 1996.
[16] Coste-Maniere E, Simon D. Reactive objects in a task level open controller. IEEE International Conference on Robotics and Automation, 1992. p. 2732–7.
[17] Rumbaugh J, Blaha M, Premerlani, Eddy WF, Lorensen W. Object-oriented modeling and design. Englewood Cliffs, NJ: Prentice-Hall, 1991.
[18] CODE, Getting started with CODE for Windows NT, Cimetrix Inc., 1996.
[19] Hwang HP, Ho CS. Knowledge task-level programming and execution environment for robots. Robotics Comput Integrated Manuf 1996;12(4):329–51.
[20] Jobling CP, Grant PW, Barker HA, Townsend P. Object-oriented programming in control system design: a survey. Automatica, 1994;30(8):1221–61.