Information Technology Journal 3 (3): 296-302, 2004
ISSN 1682-6027

A PC-based Open Architecture Controller for Robot

[1]Pan Liandong, [1]Huang Xinhan and [2]Mohammad Arif
[1]Department of Control Science and Technology,
Huazhong University of Science and Technology, Wu Han, People's Republic of China
[2]Faculty of Electrical Engineering, Pakistan Institute of Engineering and Applied Science, Pakistan

Abstract: Research on open architecture system has pointed out that a component-based philosophy should be the solution. In this study, a PC-based open architecture robot control system was presented and the technologies related to component-ware were investigated. The system is constructed based upon a standard PC platform and a particular component can be easily integrated and/or replaced. CORBA is exploited in software development to ensure the extensibility, scalability and portability of the software. Finally, by applying to the Movemaster-EX robot, the performance of the system was evaluated.

Key words: PC-based controller, robotics, open architecture, CORBA

INTRODUCTION

Researchers dealing with robots always have to face several problems due to the closed and not universal architecture of the controllers. Most of the conventional robotic controllers are designed with closed natures such as non-standard hardware interface, customized communication protocol and special program language. A closed system is hard to modify and the architecture poses great difficulties to the users to integrate external hardware and software. This desirable when the application is well defined and is not expected to change[1]. However, increased capability and flexibility is needed in many cases. This has led to the demand for controllers based on open architectures.

Although the need for a new and vendor neutral open architecture controller was emerging at many places around the world, there was no consensus on what open architecture controllers were. From the user point of view, an open architecture controller is desired for use. It should conform to industrial standards and should require modification as little as possible with respect to up-gradation or reconfiguration of hardware and software. Also, such a controller should provide a user-friendly interface for simple applications and convenient tools for experienced designers to accelerate their developments. Regardless of the details, an open architecture means qualities commonly know as "the-ilities"[2], such as flexibility, portability, scalability, reconfigurability, modularity, extendibility, reliability, interoperability and reusability[1].

Since the early years of 1980's, many projects have been carried out to develop an open architecture controller, such as NGC[3], NASREM, GISC[4], ROBLINE[5], DICAM[6], Chimera Port-based objects[7] and so on. They try to solve the problem of realization of an open architecture controller and several prototype systems have been developed. Although they are not widespread accepted due to the overly restrictive definitions and special standards, these practices have pointed out that a component-based philosophy should be the solution to the open architecture controllers.

The key of a component-based philosophy is the definition of the components interface and the basic infrastructure. In manufacturing areas, one of the most important work was done from 1992 within the frame of the European project named OSACA[8]. Similar efforts are going on in Japan named OSEC under the IROFA Consortium[9] and in the U.S. within the OMAC[10,11] projects where many earlier American research results were collected. Researchers have focused on developing an open control system including the reference model of components, the general application interface and the infrastructure that make all the components work together. As far as integration and interoperability are concerned, these approaches do not force all manufacturing vendors to conform to one special standard. Instead, these approaches use existing resources to realize user-specific functions.

It is not surprise to introduce the PC-related technologies to the development of an open architecture controller, because the PC is an example of an existing

Corresponding Author: Pan Liandong, Department of Control Science and Technology,
Huazhong University of Science and Technology, Wu Han, People's Republic of China
E-mail: pane@263.net

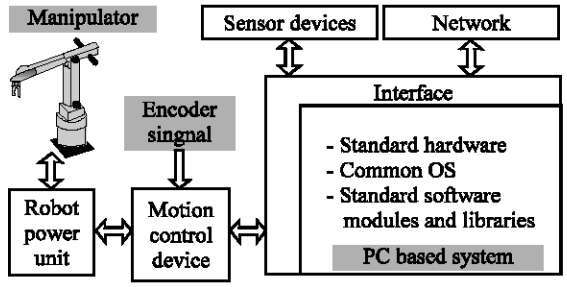


Fig. 1: Block diagram of the PC based open architecture robot counter

open architecture system that is based on the original IBM personal computer. The PC hardware architecture is now a standard piece of computing hardware that can be found in commercial products and industrial machine tools. Standard operating systems are used in millions of PC systems. Robot controller systems built around the PC and the standard operating systems can take advantage of ever-increasing power of PC hardware and software technologies.

System description: Figure 1 illustrates the block diagram of the PC-based open architecture controller. The controller includes standard PC with network connection, motion control device, power unit and external sensor devices. Also we can call it "PC Augmented Architecture" because a PC hosts the controller. It is a hierarchical system in which the motion control device implements the real time servo routine while the PC handles all the human-machine interface, high-level plan and communication chores. Sensor devices are necessary for an intelligent robotic system to get information from the environment and the network component provides the capability of distributed control. These components are integrated together through the interfaces.

Obviously, PC is the center part in this architecture. It provides a platform for hardware integration and it also provides an environment for software development. The bus interfaces and other interfaces of a PC, such as PCI, ISA, USB, RS232 and so on, are all industrial standards now. PC devices for various purposes are available in the market. For example, a system developer can get plug-in cards ranging from video to communications to special purpose like vision or motion. Common OS (operating system) like Windows or Linux is used in this system. A common OS means a de-facto standard environment for software development. By using a common OS, the developer can program with high-level languages (C, C++, Java, etc) and make use of many tools provided by third-party developers to reduce the time and cost in software development. The software of this system is a collection of functional modules. These modules can be built from scratch or may be a COTS (commercial off-the-shell) module.

One of the benefits from the use of PC based architecture is the ability to take advantage of the components from different vendors. It means that users do not rely on special vendors. In addition, the competition between vendors will result in rapid improvements in cost and performance.

Software development

Requirements: In company with the standardization of the hardware interfaces, the openness of hardware has been realized elementally. One hardware device can be connected with the corresponding devices made by another producer. So at present it is urgent to require that controlling software encompass openness. The software of an open architecture controller should have following features[12,13]:

Extensibility: A software module can be integrated into the system without affecting existing modules and a module can be replaced by another module with a same functionality.

Portability: The software of robot controller should be easily compilable to run on a variety of CPUs and operating systems, with minor or no changes in its internal structure.

Interoperability: The software modules should be able to exchange data, events and send invocations to each other regardless of the internal implementation.

CBSE (component-based software engineering) should be the way to achieve these requirements[14]. In reality, the functional modules of a robot controller are always developed by heterogeneous technologies. Some problems we have to focus on: what a component should be? How to package modules into components and how to arrange the components to form any architecture? Actually, what we need is a common framework. Some solutions have been proposed in recent years in the field of modern software engineering.

Component issue: Components here are software building blocks, each with a specific functionality that the user can assemble to achieve the global functionality of the robot. They have a standardized way to exchange information between each other so that they can be connected together.

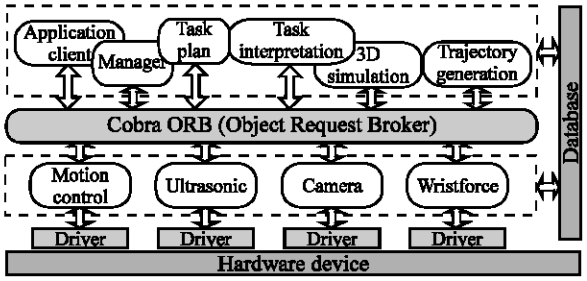


Fig. 2: The software architecture

Components are executable. They represent an encapsulation of the notion of processes and threads. They will be dedicated to a particular sensor or actuator (and handle the data of that sensor or control that actuator) and they can also provide abstract and high-level services and rely on data produced by other components.

Components execute their functionality (or deliver their services) independently of the context in which they are used, i.e. without having to know during design and implementation who is going to use its services, how they are going to be used. The independence of a component is a guide to decide what functionality should be put in it, but it is not absolute, this choice is left to the developer.

In order to accomplish tasks, the services provided by components are activated by external entities according to the task requirements and the execution context. The sequence of services can be directly activated by a human operator, planned and trigged by decisional components, or more generally by events generated elsewhere in the system. It is important to stress that this structuring does not presuppose any decisional architecture: the way components are connected together is not forced nor specified. Hence, these controllable components can be used for implementing any decisional architecture, form behavior based to supervised and planned approaches purposes, when reused in different architectures.

Component technologies: At present, there are three mainstream component technologies[15]. These are Common Object Request Broker Architecture (CORBA), Enterprise Java Beans (EJB) and Component Object Model (COM). EJB is suitable when the system purely develop by Java. COM is mainly intended to be used in Microsoft based environment. The CORBA specification is the result of input of a wide range of OMG[16] members, making its implementations the most generally applicable option. CORBA gives application components the ability to transparently interoperate with objects implemented on any of the more than 50 platforms with CORBA support, including all common Windows, Unix and Linux variants. CORBA also provides seamless communication between components implemented in different programming language.

The central part of CORBA is the Object Request Broker (ORB). It encompasses the entire communication infrastructure necessary to identify and locate objects, handle connection management and deliver data. Another fundamental piece of the CORBA architecture is the use of the Interface Definition Language (IDL) to ensure CORBA's language independence. And the General Inter-ORB Protocol (GIOP) ensures true interoperability among components. The CORBA Component Model (CCM) proposed in CORBA 3.0 make it more suitable for component-based software development. CCM uses a container to host the components and manage system services implicitly. The component implementations can only contain the application business logic. Detail information of CORBA is described in OMG[16-18].

Software architecture: The CORBA-based software architecture of robot controller was presented in Fig. 2. The components defined in this architecture could be divided in two categories: Device components (Motion Control, Ultrasonic, Camera, Wrist Force) and Logic components (Application Client, Manager, Task Plan, Task Interpretation, Trajectory Generation, 3D Simulation). Device component are abstractions of the hardware devices. They deal with the physical devices through the drivers and hide the details of the actual physical device for the other components. Device manufacturer always provides the driver. But if you develop your own hardware, you have to write it yourself. Logic components are not dedicated to physical devices. They are designed to provide some logical services such as kinematics, dynamics, fusion algorithm or some manage functions and so on. All the components connected together through the ORB.

Most of the components in the architecture can be reused in other robot control systems. Their names imply the functions. Task Plan component is responsible for generating the sequence of operations according to the task requirement. Plan Interpretation component reads and executes the sequences and produces instructions in code level. The instructions are send to the Trajectory Generation component, the job of this component is to plan and execute coordinated axis motion, such as linearly interpolated moves or circular arcs. The Trajectory Generation component computes a sequence of positions (plus velocities or other quantities, depending on the requirement) to the Motion Control component, which converts them to axis positions and issues control signals

to the motors. Different control laws can be configured, such as PID, pole placement or gain scheduling. Camera, Ultrasonic and Wrist Force components, as mentioned above, deal with the corresponding sensor devices. The ultrasonic sensor and wrist force sensor are developed by our lab, so the drivers are written by ourselves. The Manager component manages the other components. It is the first component that should be executed in this system. It is responsible for registering, activating and deactivating the components. The Application Client component is a special module developed by the user according to the application environment. Its job is to define the logic decisional architecture of the controller and assemble the components together to complete the task. It is started by the Manager component and should be in charge of the execution of the whole software system. Finally, the 3D simulation components can provide some simulation functions and demonstrate the results in 3D virtual environment. Each of the components has its own graphic interface to provide convenient monitoring and setting. Also, a database is implemented to store the related information in the system, such as sensory data, robot type, device information, system environment and so on.

Component development: We develop the components in accordance with the CORBA 3.0 specification. Stages of component development lifecycle can be classified as follows: interface design, component design, component implementation, component packaging, application assembly and system deployment.

Many CORBA-compliant products have been developed to support the component-based software development. Some of them are free, such as CIAO by Washington University and Vanderbilt University[19,20], EJCCM by Computational Physics Inc[21]. MICOCMM by Frank Pilhofer[22] and Open CCM by Object Web[23]. These tools free us from many nitty-gritty details of component software integration and interoperability, so attention can be focused on interface design (specification of supported interfaces of components). Following code segment demonstrates an interface defined in the 3D simulation component used IDL. Figure 3 shows a screen-shot of the software.

```
typedef sequence<float> posedata;
//Interface definition for kinematics simulation
Interface Kinematics {
    attribute short RobotType;
    attribute short SimMode;
    void SetStartPose( in posedata StartPose);
    void SetEndPose( in posedata EndPose);
    void Start();
                …
    };
```

Fig. 3: A screen-shot of the 3D simulation

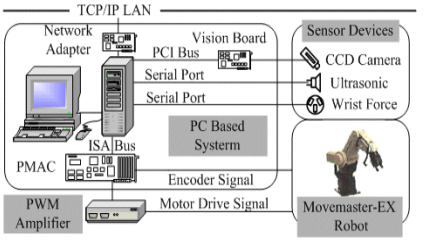


Fig. 4: The experimental platform

Experimental system and results: We apply the developed system to a Movemaster-EX robot and replace its original proprietary controller. Figure 4 depicts the hardware structure of the system which includes a PC, a PMAC motion controller, a PWM amplifier and some sensory devices. There are three sensors used in this system, a hand-mounted CCD camera, a 3-channel ultrasonic sensor and a force/torque sensor installed on wrist of the robot. The PC is running the Windows 2000 operating system. We use VC++ 6.0 to develop the program and take advantage of some software packages (CIAO).

The capability of the PC-based controller can be evaluated from two different aspects. The first aspect was to examine how easy system integrations and modifications. The second aspect was to examine the performance of the control. Actually the first aspect is obvious. The PC-based standard interface hardware and the component-based software ensure that the extensibility and scalability are available. The hardware components and software components can be integrated or replaced easily. Following experiments give an overview of performance of the control system.

Trajectory tracking: Firstly evaluated the motion control of this system. In this experiment, step move and parabolic move (cubic in position, parabolic in velocity, linear varying in acceleration) are commanded to the waist joint (first axis) of the robot and the command trajectory and output trajectory are compared to illustrate the response performance. The servo control law is PID plus feedforward (Fig. 5). Figure 6a is a response

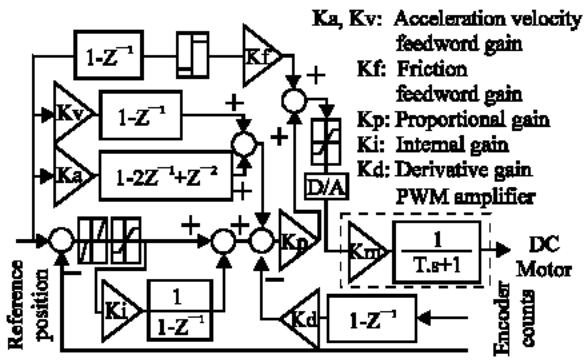


Fig. 5: The servo control scheme

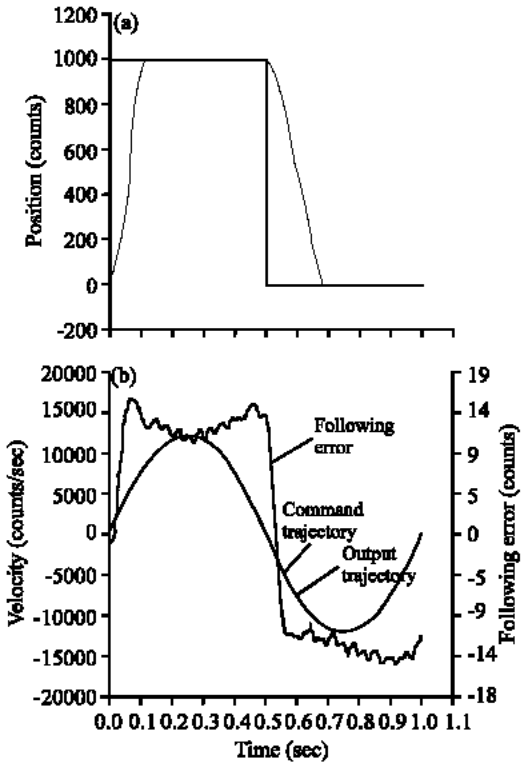


Fig. 6: Tracking performance of first axis (waist) of movemaster-EX

of step input. Figure 6b illustrates the tracking performance for parabolic input. The unit of the Y-axis is encoder counts. In Fig. 6b, it is very difficult to distinguish the command trajectory and output trajectory because the small difference between them. The results demonstrated that the performance of real-time motion control is satisfactory.

Measurement of work-pieces' profile size and 3D reconstruction: On-line profile measurement is important for assembling task. This experiment demonstrated the

(a) Robot pose (b) Work piece1 (c) Work piece2

Fig. 7: Images of robot pose and work-pieces

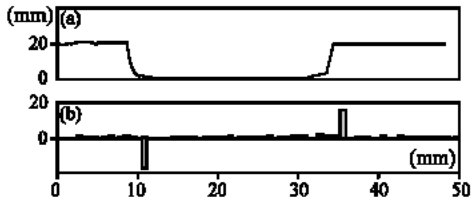
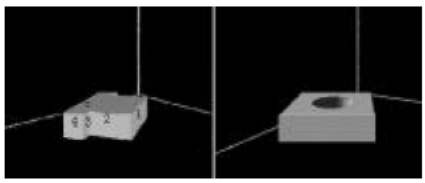


Fig. 8: Output of depth detection
(a) Output curve of ultrasonic sensor (b) Differential result



(a) Work piece1 (b) Work piece 2

Fig. 9: Results of 3D reconstruction

cooperation between the components. Sensors are manipulated to get information needed to complete the task. In this experiment, the CCD camera and the ultrasonic sensor are used to conduct the non-contact measurement of the work-piece profile size. The size information will send to the 3D Simulation component and the work-piece will be reconstructed in the virtual environment. To simplify the task, firstly the robot moves to a predefined pose as illustrated in Fig. 7a (the end effecter is vertical to the work table). And then the work-piece is placed in view of the hand-mounted CCD camera. So the system has no need to search in a large-scale area to locate the work-piece. The task is executed as follows:

I. Get location information of the work-piece and centralize it in the view of the camera through movement of the robot.
II. Process the images (threshold selecting, edge extraction, corner detection) and get geometric information of the work-piece, such as shape, midpoint, length and orientation.
III. Take advantage of the information obtained in II and guide the ultrasonic sensor to detect the depth information.

Table 1: Profile size of work-piece 1 (SL=side length, the index numbers of sides are labeled in Fig. 9a)

Unit (mm)	SL_1	SL_2	SL_3
True value	59.1	40.2	10.2
Measured value	59.2	39.8	10.6
Error	-0.1	0.4	-0.4
Unit (mm)	SL_4	SL_5	Height
True value	19.9	79.2	20.7
Measured value	19.8	79.3	20.6
Error	0.1	-0.1	0.1

Table 2: Profile size of work-piece 2 (SL=side length)

Unit (mm)	SL	Diameter	Height
True value	79.1	40.8	19.9
Measured value	79.3	40.6	20.0
Error	-0.2	0.2	-0.1

The work-pieces used in this experiment are simple and regular. For complex work-pieces, the image process algorithm and depth detection plan should be more complex. Two work-pieces are used. One is a "T" shaped piece and the other is a cuboid with a hole in it as shown in Fig. 7b and 7c. To get the depth information, the ultrasonic will scan through the whole work-piece. Several scan lines are planed based on the information obtained in procedure II. Figure 8 demonstrated the output of the ultrasonic sensor. The sensor conducts detection every 1 mm along the scan line. Part of the output curve is showed in Fig. 8a. Figure 8b is the differential result of Fig. 8a. The depth information can be obtained by analyzing Fig. 8b. Compensation values should be added to the results especially in the edge points because of the mechanical random vibration and ultrasonic diffusion. The experimental results are listed in Table 1 and 2. These data are sent to the 3D Simulation and the reconstruction results are shown in Fig. 9.

Conclusions and future work: This study presented a PC-based open architecture controller for robot. We did some work for bringing robotics into the component philosophy. By assembling controller from off-the-shell hardware and software components, the benefits of reduced cost and improved robustness have been realized. Also the controller is able to provide some characteristics that an open architecture system should have. To achieve these, some technologies that have been available from computer science are exploited: standard PC architecture and interface, common OS, component-ware technology like CORBA and high-level programming language.

The system has been designed as a basic on which to implement more complex control applications. Currently we are working on some extensions:

Implementing a hard real-time system. Many robotic tasks have critical time requirement, they cannot be completed on a Not Real-time (NRT) OS like Windows. We should implement the system on a RTOS (RT-Linux, QNX, OS9, etc). And the features of Real-time CORBA should be explored and used to enhance the system[24].

Implementing the system in distributed manner. We have to consider the distributed requirement for either reason as follow. One is that the elements of the robotic system are distributed in different location. Another is that tasks could be completed more efficient by assigned to several computers connected by network. The use of CORBA makes it easier than before.

REFERENCES

1. William, E.F., 1994. What is an Open Architecture Robot Controller? IEEE International Symposium on Intelligent Control, Columbus, Ohio, USA., pp: 27-32.
2. Frederick, M.P. and J.S. Albus, 1997. Open-Architecture Controllers. IEEE Spectrum, pp: 60-64.
3. Leahy, M.B.Jr. and S.B. Petroski, 1994. Unified Telerobotic Architecture Project Status Report. IEEE International Conference on Systems, Man and Cybernetics, Oct. 1994, 1: 249-253.
4. Miller, D.J. and R.C. Lennox, 1991. An Object-Oriented Environment for Robot System Architectures. Control Systems Magazine, IEEE, Feb. 1991, 11: 14-23.
5. Sorensen, S., 1993. Overview of a modular, industry standard based open architecture controller. Proceedings of the International Robots and Vision Automation Conf., Detroit, April 1993.
6. Hayes-Roth, F., L.D. Erman, A. Terry and B. Hayes-Roth, 1992. Domain-specific Software Architectures: Distributed Intelligent Control and Management. IEEE Symposium on Computer-Aided Control System Design, March 1992, pp: 117-128.
7. Stewart, D.B., R.A. Volpe and P.K. Khosla, 1997. Design of Dynamically Reconfigurable Real-time Software Using Port-based Objects. IEEE Transactions on Software Engineering, 23: 759-776.
8. OSACA., 1996. Open system architecture for controls within automation systems. ESPRIT III Project 6379/9115 Final Report. http://www.osaca.org.
9. Sawada, C. and O. Akira, 1997. Open controller architecture OSEC-II: Architecture overview and prototype systems. Proceedings of 6th International Conference on Emerging Technologies and Factory Automation, Sept. 1997, pp: 543-550.
10. MAC., 2004. Open, modular architecture control. http://www.omac.org.
11. Kolla, S., J. Michaloski and W. Rippey, 2002. Evaluation of component-based reconfigurable machine controllers. Proceedings of the 5th Biannual World Automation Congress, June 2002, 14: 625-630.

12. Frenandez, J.A. and J. Gonzalez, 1999. The NEXUS open system for integrating robotic software. Robotics Computer Integrated Manufacturing, 15: 431-440.
13. Hua, Xu, P.F. Jia, M. Yang and H.Z. Liu, 2002. Portability in General Open Controllers, TENCON '02. Proceedings, Oct. 2002, 3: 1574-1577.
14. George, T.H. and W.T. Councill, 2001. Component-Based Software Engineering: Putting the Pieces Together. Addison Wesley.
15. Emmerich, W. and N. Kaveh, 2002. Component technologies: Java Beans, COM, CORBA, RMI, EJB and the CORBA component model. Proceedings of the 24th International Conference on Software Engineering, May 2002, pp: 691-692.
16. OMG., 2004. Object management group. http://www.omg.org.
17. OMG., 2004. The Common Object Request Broker: Architecture and Specification. Version 3.0.3-Editorial changes formal/04-03-01, Object Management Group, March 2004.
18. OMG., 2002. CORBA Components, Version 3.0 formal/02-06-65. June 2002.
19. CIAO., 2004. Component Integrated ACE (Adaptive Communication Environment) ORB (Object Request Broker). http://www.dre.Vanderbilt.edu/CIAO.
20. Wang, N. and C. Gill, 2004. Improving real-time system configuration via a qos-aware CORBA component model. Proceedings of the 37th Annual Hawaii International Conference on System Sciences, Jan. 2004, pp: 273-282.
21. EJCCM., 2004. Enterprise java CORBA component Model. http://www.cpi.com/ejccm.
22. MICOCMM., 2004. The MICO CORBA component Project. http://www.fpx.de/MicoCCM.
23. OpenCCM., 2004. The open CORBA Component model platform. http://openccm.objectweb.org.
24. Sanz, R., M. Alonso, I. Lopez and C.A. Garcia, 2001. Enhancing control architecture using CORBA. Proceedings of the 2001 IEEE International Symposium on Intelligent Control, Sept. 2001, pp: 189-194.