2014 IEEE International Conference on Robotics & Automation (ICRA)
Hong Kong Convention and Exhibition Center
May 31 - June 7, 2014. Hong Kong, China

RGMP-ROS: a Real-time ROS Architecture of Hybrid RTOS and GPOS on Multi-core Processor

Hongxing Wei, *Member, IEEE,* Zhen Huang, Qiang Yu, Miao Liu, Yong Guan，and Jindong Tan, *Member, IEEE*

***Abstract*—Recently, the open-source robot operating system (ROS) has been growing rapidly in the robotics community. However, the ROS runs on Linux, which does not provide timing guarantees for robot motion. This paper present a hybrid real-time ROS architecture on multi-core processor "RGMP-ROS", which consists of two parts including the non-real-time subsystem "GPOS (General Operating system)" and the real-time one "RTOS (Real-time Operating system)". The GPOS is comprised of non-real-time ROS nodes running in Linux, while the RTOS only contains real-time ROS nodes running in Nuttx. To get higher operational efficiency, the RGMP-ROS system is executed by a dual-core processor, one CPU for GPOS and the other for RTOS. The RGMP-ROS has used in the controller of a 6-DOF modular manipulator, and its effectiveness and efficiency are demonstrated by software testing and experiments. The main contributions of the present work lie in the realization of real-time ROS architecture and the application of multi-core processor in the hybrid control of an industrial robot.**

I. Introduction

With the development of robotics, two problems in this field have been widely concerned in these years. The first is about the enhancement of the development efficiency of the controlling software. As is known, most existing industrial robots are designed to perform special tasks in specific environments. Because different robots generally have different controlling software, developers always have to spend a lot of time writing diverse programs with similar functions to meet new operational requirements frequently. Therefore, the complexity of programming remains as one of the major hurdles in this field [1]. The other problem is about the enhancement of the operational efficiency of robot control. Such efficiency mainly depends on the execution speed of the controlling software. An effective way to solve the first problem is to construct modular software architecture. By dividing an application into a number of mutually decoupled units, the complexity of the software development can be reduced to a large extent. The designers only need to update the corresponding software modules, when changing the functionalities or adding features to the robots. Because most of the modules are reusable and portable for similar operational tasks, the adaptation of the controlling software can be accomplished quickly. Although the modularization of software is much more difficult than that of hardware, some modular frameworks have been successfully applied in the controlling software of robots such as CARMEN [2], Player [3], LCM [4], YARP [5], and etc. Some surveys on robot middleware and robot development environment are given in [6] and [7].

As mentioned above, the operational efficiency of industrial robots is mainly determined by the running speed of the controlling software. There are several ways to speed up the software system. Firstly, a number of microcontroller modules can be connected together to form a message-based [8] or event-based [9] distributed control system for the robot, in which the controller nodes communicate with one another through a field bus [10]. Secondly, the real-time operating system and the non-real-time one can be combined together to construct a hybrid one [11], which can handle both real-time and non-real-time tasks simultaneously by separating the corresponding interrupts automatically. Thirdly, the multi-core processors can be employed to form a distributed system [12], and thus the real-time and non-real-time software can run in different CPUs of the same processor. Conventional systems are generally based on two-level single-core processors with the low-level processor for the real-time software and the high-level ones for the non-real-time software [13]. Because a considerable amount of time is consumed by the frequent communication between different processors, the operational efficiency of the whole system is inevitably lowered down. Obviously, such insufficiency of the conventional two-level systems can be overcome effectively by the multi-core method. To summarize, the modular distributed architecture, the hybrid operating system and the multi-core processors are three main techniques available for the enhancement of the operational efficiency of industrial robots.

Recently, the open-source robot operating system (ROS) has been growing rapidly in the robotics community [14]. Based on the ROS framework, many researchers have developed their software for diverse robots such as Barrett WAM [15] and Raven-II [16]. However, ROS runs on Linux, which does not provide timing guarantees for robot motion.

Hongxing Wei and Zhen Huang are with the School of Mechanical Engineering and Automation, Beihang University, Beijing 100191, PR China (e-mail: weihongxing@buaa.edu.cn).

Qiang Yu and Miao Liu are with the Shanghai Institute of Microsystem and Information Technology, Shanghai 200050, China (e-mail: yuq825@gmail.com and threewater1@163.com).

Yong Guan is with College of Information Engineering, Capital Normal University, Beijing, 100048, PR China (e-mail: guanyxxxy@263.net)

Jindong Tan is with the Department of Mechanical, Aerospace, and Biomedical Engineering, The University of Tennessee, Knoxville, Tennessee 37996-2110, USA (e-mail: tan@utk.edu)

The need for timing guarantees drives robot designers to partition robots into real-time and non real-time subsystems. In order to meet real-time needs, there are two possible approach. The first is attaching embedded real-time systems to ROS, such as ROS *Industrial* and ROS *Bridge*. A second approach is to port some ROS packages to a real-time Unix version [17].

Unlike the above approach, this paper present a real-time ROS architecture called RGMP-ROS, which is based on the hybrid OS platform RGMP (RTOS and GPOS on Multi-Processors) developed by our research team [18]. Here, we chose an open source Nuttx [19]. To facilitate the operation, a dual-core processor is used to run the whole software system. The real-time and the non-real-time subsystems of RGMP-ROS are executed in different CPUs of the dual-core processor, respectively. To verify its effectiveness, the RGMP-ROS system is further applied in the industrial controller of 6-DOF modular manipulator designed by the authors, and the feasibility and stability of the whole software system is finally demonstrated by the testing experiments. The main contributions of the present work lie in two aspects: (a) the real-time control functionality is integrated into the ROS-based hybrid operating system; (b) a dual-core processor is employed to run the hybrid operating system, one core for the real-time part and the other for the non-real-time part.

The remainder of this paper is organized as follows. Section Ⅱ introduces the technical backgrounds of ROS, Nuttx and RGMP. Section Ⅲ presents the design and implementation of RGMP-ROS. Software testing is then conducted in section Ⅳ for the purpose of performance verification. In section Ⅴ, the RGMP-ROS system is further applied in the controller of a six-DOF modular manipulator, and some experiments are done to demonstrate its feasibility. Finally, the conclusions are drawn in section Ⅵ.

II. TECHNICAL BACKGROUNDS

A. ROS

ROS (Robot Operating System) is an open-source and reusable software platform providing libraries, tools and conventions that can help to create high-performance robot applications quickly and easily. It provides the international robotics community with standardized interfaces to the hardware, tools for creating, debugging, distributing, and running procedures, and libraries for developing programs. Now, about 500 packages have been made available in the ROS setting by approximately 30 institutions [20]-[22].

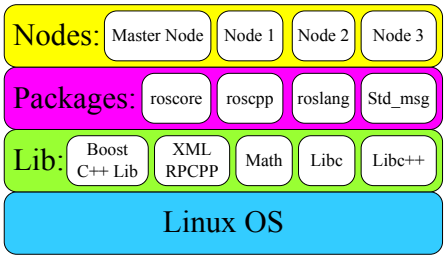


Fig. 1. The ROS software architecture

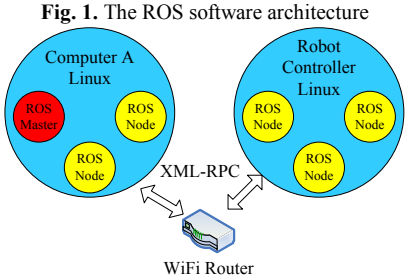


Fig. 2. The remote control model of ROS

ROS is a modular software platform installed on the Linux operating system. It contains a number of software modules encapsulated as nodes, including the master node and the functional nodes. Fig. 1 gives the standard architecture of the ROS software system, which consists of libraries, packages and nodes installed on Linux OS. Such a modular architecture supports distributed network communication. Like the router in a LAN, the master node administrates and monitors the running of the functional nodes and their peer-to-peer communications. Fig. 2 shows a typical remote control model of ROS. The ROS in the robot is connected to that in the remote control computer through the WIFI router. The communications among nodes are realized by the XML/RPC calls under the TCP/IP protocol. Fig. 3 illustrates the ROS communication model and its corresponding OSI model. It is seen that an application layer based on the XML-RPC HTTP protocol is constructed on the TCP/IP architecture. Therefore, the messages transmitted by the nodes are not data packages but webpage files satisfying the http protocol.

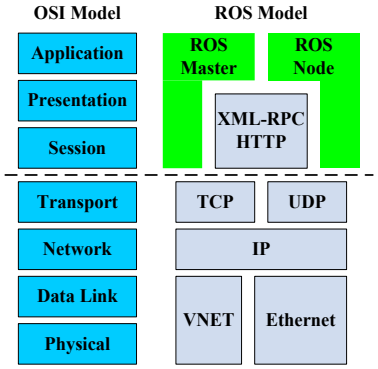


Fig. 3. The ROS communication model and its corresponding OSI model

B. Nuttx

In order to construct the hybrid operating system, we need a portable real-time operating system architecture. As we all know, there are many real-time operating system, such as VxWorks, QNX, eCOS, µcOS-II, etc. However, we want to choose a open-source RTOS to easily extend ROS. Due to this reason, Nuttx is chosen as the prototype of our RTOS.

Nuttx is an open-source embedded real time operating system (RTOS) first released in 2007 by Gregory Nutt [19]. Nuttx supports the Posix and ANSI standards, and can be applied in microcontrollers from 8-bit to 32-bit. In addition,

by adopting the standard APIs from Unix and other common RTOS's such as VxWorks, Nuttx has some functionalities not available under the Posix and ANSI standards. Compared with other real-time operating systems such as VxWorks, Windows CE and μC/OS-II, Nuttx has the following advantages:

(1) Practical system services. Nuttx provides a number of system supports such as UIP protocol stack, networking, device drivers and virtue file system, which are useful for practical systems.

(2) Small footprint. Nuttx only has very small memory requirement that can be met in any application cases. For example, the 4MB memory released by the GPOS of our hybrid OS is enough to run our Nuttx-based RTOS.

(3) Easy extension. It is convenient to extend Nuttx to new processor architectures such as the SoC architecture and the board architectures. This makes it possible to transplant ROS real-time nodes between different CPUs of a multi-core processor. Therefore, in our hybrid OS, the ROS real-time nodes run in the Nuttx as applications of RTOS.

C. RGMP

RGMP is a software framework developed by our research team [18], whose architecture is shown in Fig. 4. It can be used to run two operating systems simultaneously in a controller having a multi-core processor. One is the Linux-based general purpose operating system (GPOS), and the other is the real time operating system (RTOS). GPOS and RTOS occupy different CPUs, respectively, and they communicate with each other through the VNET channel. VNET is a virtual network interface and interacts with the underlying data by sharing memory. Meanwhile, standard network interface protocols are used to establish standard network communication interface based on TCP/IP protocol between Nuttx real-time system and Linux.

RGMP has three functionalities. The first one is to boot the whole system. When RGMP starts, Linux initializes all the hardware resources including the CPUs, the memory and the peripheral devices at first. Then, the GPOS module releases two blocks of memory and a CPU. After this, the RTOS is loaded into one block of the released memory and executed by the released CPU. The second functionality of RGMP is to realize the communication between GPOS and RTOS during the running process. Such communication is performed in the other released memory block. The last but not the least

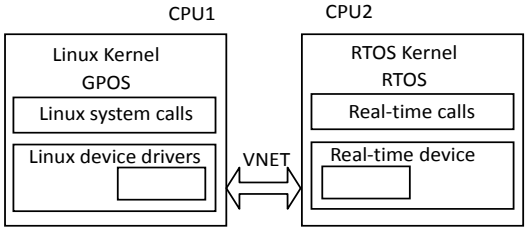


Fig. 4. The software architecture of RGMP

functionality of RGMP is to provide a great amount of programming resources such as the interface functions for interrupt registration, memory allocation, BIOS reading, etc. With these resources, the users can develop their own hybrid operating system quickly.

III. Design and Implementation of RGMP-ROS

A. Overall design

RGMP-ROS is developed by using ROS, Nuttx and RGMP. As illustrated in Fig. 5, RGMP-ROS is a hybrid operating system running in a dual-core processor and composed of two parts. One part is the standard non-real-time ROS system (i.e., GPOS) installed in Linux, and the other part is the real-time ROS system (i.e., RTOS) running in Nuttx. Each part has its own CPU, memory, interrupts and peripheral devices. Accordingly, there are two kinds of ROS nodes, i.e., the real-time nodes and the non-real-time ones. The communications among nodes are performed by the XML/RPC calls under the TCP/IP protocol. Because the real-time ROS and the non-real-time ROS run in different CPUs, the RGMP-VNET acts as the channel for their communications.

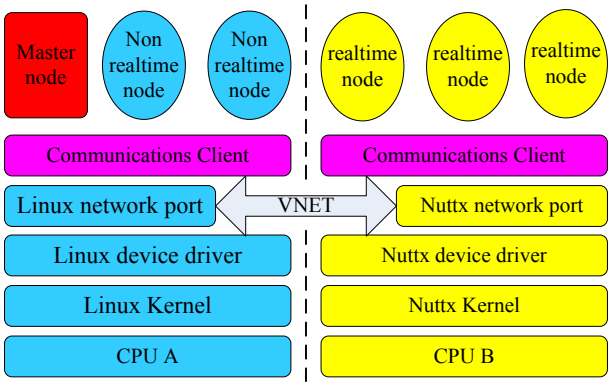


Fig. 5. The RGMP-ROS architecture

B. Implementation

RGMP-ROS has four software modules including ROS, RGMP, Nuttx and RTroscpp, which are developed in personal computer. The developments of these modules require the supports of the following software: Linux, ROS, Git, Qemu, Vim (or Emacs), Gcc, Gdb, Ctages, Eclipse, and the source codes of RGMP, Linux and Nuttx. Besides, we have to add some special software packages to Nuttx for our program development, although Nuttx has rich resources itself. Firstly, in order to develop the real-time ROS nodes such as the RTroscpp nodes, the C++ support needs to be added. Here, the light-weight library μClibc++ is inserted into Nuttx for the purpose of developing light-weight RTOS. Secondly, the floating-point math library in standard libc should be appended, due to the fact that floating-point calculation is always requisite in the real-time control of robots. Thirdly, the http protocol needs to be transplanted, because the messages transmitted in the communications are special http webpage files not supported by the UIP protocol stack of

Nuttx. Fourthly, the drivers of the real-time PCI, Ethercat and CAN bus devices are also added to Nuttx, since each real-time bus has a separate protocol framework.

IV. SOFTWARE TESTING

An X86-based industrial robot dual-core controller is employed to test the overall performance of RGMP-ROS, including the independency between GPOS and RTOS, and the execution efficiency of the whole system. The testing platform consists of the following hardware: (1) mainboard: AIMB-780; (2) CPU：Intel(R) Pentium (R) Dual CPU E2200 of 2.2GHz; (3) Memory: DDR2 of 2GB; (4) CAN device: Advantech PCI-1680U CAN; (5) VGA card: HD6670 DDR5 of 1GB. The software of the testing platform includes: (1) Ubuntu 11.10 with Linux Kernel 3.2; (2) the fully installed ROS future; (3) OpenGL v1.4; (4) RGMP v4.3; (5) Nuttx v6.26 with μClibc++, math lib and real-time device drivers; (6) Matlab 2010a.

A. *CPU occupancy testing*

The CPU occupation of the dual-core processor is tested and compared in two cases. In the first case, we run the ROS core node and all the non-real-time ROS nodes in Linux, but the real-time system in Nuttx is not started. The CPU occupation result is shown by the system resource manager (see Fig. 6), which indicates that the GPOS tasks are shared by both CPUs. The occupancy rate of CPU1 is 52.9%, and that of CPU2 is 20.4%. In the second case, we run the non-real-time ROS nodes in Linux and the real-time ROS nodes in Nuttx at the same time. The system resource manager in Linux then displays that the occupancy rate of CPU2 becomes zero (see Fig. 7), which shows that CPU2 is totally occupied by the Nuttx real-time system. Therefore, GPOS and RTOS can run simultaneously, each occupying a separate CPU.

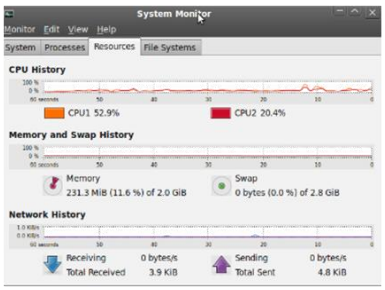


Fig. 6. CPU occupancy rates in case

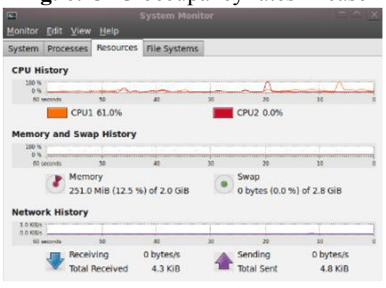
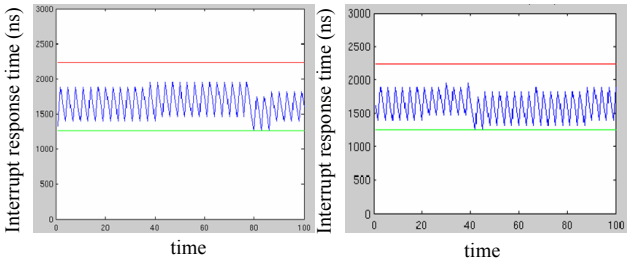


Fig. 7. CPU occupancy rates in case 2

B. *Real-time interrupt response time testing*

An important index to evaluate the performance of the real-time system is the interrupt response time. To test it exactly, we only run a single real-time ROS node in Nuttx, which handles the external timer interrupt. Two different cases are surveyed to show the effects of the non-real-time nodes on the real-time interrupt response time (see Fig. 8). In case A, the real-time ROS node runs in Nuttx when only the ROS master node is running simultaneously in Linux; while in case B, the real-time ROS node runs with a 3D RVIZ simulation program running meanwhile in Linux. For the convenience of displaying, a Matlab script is written to show the interrupt response time dynamically. The testing results are given in Fig. 8, where the upper straight line denotes the highest value of the interrupt response time, while the lower one indicates the lowest value. It is seen that the interrupt response time in both cases lies in the same interval from 1250ns to 2250ns, implying that the effect of the non-real-time system on the real-time one is negligible. This again verifies the independency between these two subsystems.



Case A **Case B**

Fig. 8. The real-time interrupt response time

C. *Communication time testing*

Here, the term “communication time” refers to the response time between different ROS nodes in communication. The communication time between non-real-time ROS nodes can be tested by using the standard ROS tool “rosping”, and that between two real-time ROS nodes may be measured simply by the system timer. To simplify the testing, only a single real-time ROS node is running in Nuttx, and only the roscore and the non-real-time ROS node “helloworld” for message printing are running in Linux. Because the communications among ROS nodes are made by the mechanism of XML-RPC call under the TCP protocol, the communication time is generally not fixed.

Table 1. The average communication time

Communication types	Time
Non-real-time node ↔ Non-real-time node	>1ms
Non-real-time node ↔ Real-time node	0.1~1ms
Real-time node ↔ Real-time node	<100ns

The average communication time is presented in Table 1 for three different kinds of node communications. It is shown that the average communication time between two

non-real-time nodes is longer than 1ms, that between a non-real-time node and a real-time one is about 0.1ms to 1ms, and that between two real-time nodes is less than 100ns. Therefore, the average communication time of the ROS nodes is acceptable for the controllers of practical industrial robots.

V. EXPERIMENTS

In this section, the RGMP-ROS system is applied in the controller of a 6-DOF modular manipulator developed by our research team. The hardware of the controller is the same as that of the testing platform given in section Ⅳ. The software architecture is shown in Fig. 9. For the convenience of communication and control, an ID number is assigned to each module of the manipulator. When the controller emits an instruction containing an ID number, only the module having the same ID number will receive and respond to it. If the ID number of the instruction is 255 (i.e., the hexadecimal 0xFF), it is a broadcast instruction that will be received by all the robotic modules with no response.

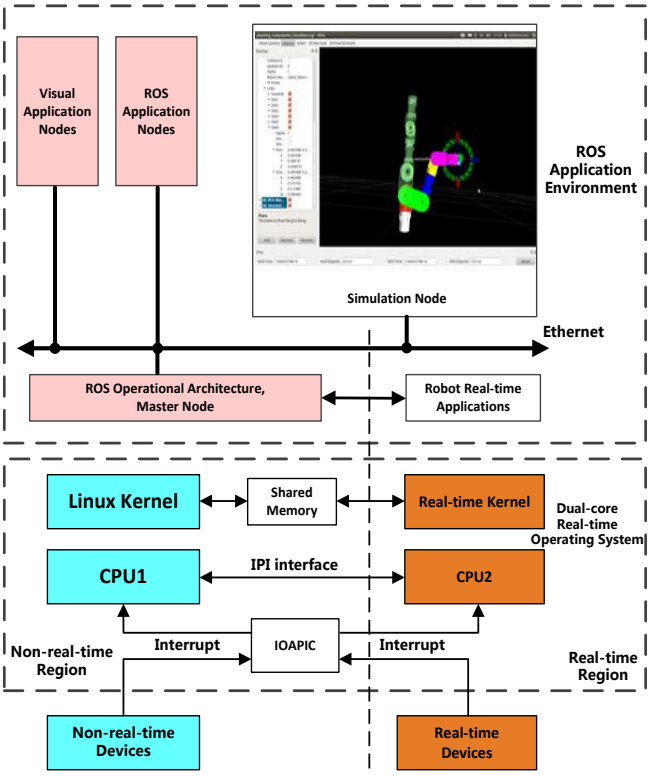
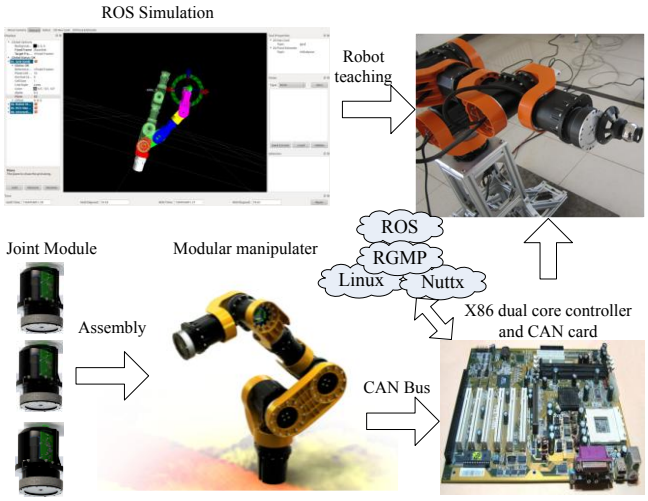


Fig. 9. The software architecture of the modular manipulator. The online teaching control is visualized in the ROS 3D demonstration platform "RVIZ". The model of the manipulator can be directly loaded from Solidworks into RVIZ to perform 3D simulation. In RVIZ, the end module of the 3D simulation model can be dragged by using the mouse to carry out the online teaching. As shown in Fig. 11, the path planning can also be realized for the manipulator in such a 3D simulation way.



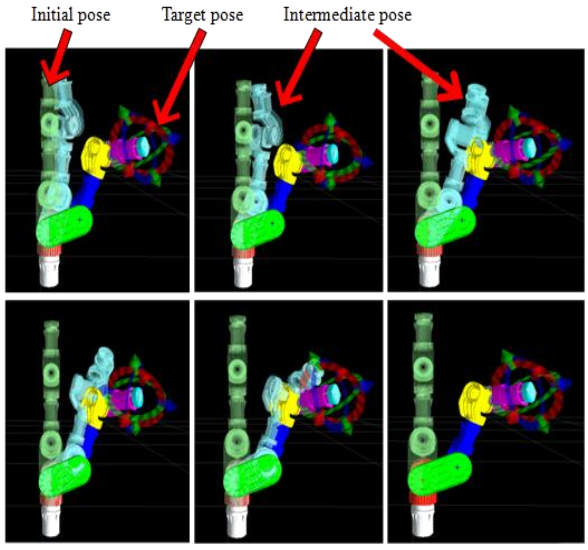


Fig.11. Path planning in RVIZ

Fig. 10. The overall control architecture of the modular manipulator

A. *Visualized teaching control*

The RGMP-ROS system is installed into both the manipulator controller and a remote computer, which are connected through Ethernet. Real-time control can be realized by running the real-time ROS nodes in Nuttx, while the remote manipulation and online teaching can be performed by running the non-real-time ROS nodes in Linux. Fig. 10 illustrates the overall control architecture of the

B. *Motion control*

The real-time ROS system performs real-time operations on the manipulator, including the closed-loop motion control, the kinematics solution, the status inquiry, the error handling, etc. During the operation process of the manipulator, anyone of the angular velocity, angular displacement (or position) and electric current can be used to control the rotation of the joints. Here, the position based method is employed, and so the

corresponding angular displacement information is emitted to every joint to limit its motion range. It is found by motion control experiments that synchronous movement among the joints can not be well guaranteed by the position based method directly. Besides the method itself, there are several other factors affecting the motion synchronism, such as the different reduction gear ratios of the joints and the friction. Theoretically speaking, to reduce the friction, to adjust the reduction gear ratios, or even to replace the position based method by the angular velocity or electric current based method, can be helpful to realize the motion synchronism. However, all these three ways may bring about unpredictable difficulties. Instead, we try a simpler and more feasible way here. A refined kinematics ROS node is developed to perform high-frequency interpolation calculation to yield more exact control information for the angular displacement. As shown in Table 2, the motion synchronism of the six joints is well guaranteed by this way.

Table 2. The time for the joints to rotate one degree angle

Joints	Time (ms)	
	The position based control method	The refined interpolation calculation method
1	350	110
2	330	108
3	331	110
4	341	109
5	320	100
6	310	105

VI. CONCLUSIONS

A hybrid real-time ROS architecture called RGMP-ROS is developed by using the libraries and tools provided by ROS and the hybrid OS platform RGMP. It consists of two parts including the non-real-time GPOS and the real-time RTOS, and accordingly there are two kinds of ROS nodes, i.e., the non-real-time ROS nodes running in Linux and the real-time ROS nodes running in Nuttx. In order to enhance the operational efficiency, a dual-core processor, instead of the conventional two-level single-core processors, is used to execute the RGMP-ROS system with one CPU for the non-real-time nodes and the other for the real-time nodes. The RGMP-ROS system is further applied in the industrial controller of a 6-DOF modular manipulator designed by our research team. Finally, the effectiveness of the RGMP-ROS system is verified by software testing and experiments.

In future, much more testing and improvement still need to be conducted to enhance the comprehensive performance of the software system. For example, the angular velocity or electric current based method can be tried to get a more exact synchronous motion control. In due time, we will upload the software system to the web of ROS for more extensive testing and applications by the worldwide robotics community.

ACKNOWLEDGMENTS

This work was supported by the National High Technology Research and Development Program of China ("863" Program) (2012AA041402 and 2012AA041405), National Natural Science Foundation of China (Grant No. 61175079 and No. 51105012).

REFERENCES

[1] Z. X. Pan, J. Polden, N. Larkin, S. V. Duin, and J. Norrish, "Recent progress on programming methods for industrial robots," *Robot. Comp.-Int. Manuf.,* vol. 28, no. 2, pp. 87–94, 2012.

[2] M. Montemerlo, N. Roy, and S. Thrun, "Perspectives on standardization in mobile robot programming: The carnegie mellon navigation (CARMEN) toolkit," *in Proc. of the IROS*, pp. 2436–2441, 2003.

[3] B. P. Gerkey, R. T. Vaughan, and A. Howard, "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems," *in Proc. of the 11th Int. Conf. on Advanced Robotics*, pp. 317–323, 2003.

[4] A. Huang, E. Olson, and D. Moore, "LCM: Lightweight communications and marshalling," *in Proc. of the IROS*, pp. 4057–4062, 2010.

[5] G. Metta, P. Fitzpatrick, and L. Natale, "YARP: Yet another robot platform," *Int. Journal on Advanced Robotics Systems*, pp. 43–48, 2006.

[6] A. Elkady and T. M. Sobh, "Robotics Middleware: A Comprehensive Literature Survey and Attribute-Based Bibliography," *Journal of Robotics*, 2012.

[7] J. Kramer and M. Scheutz, "Development environments for autonomous mobile robots: A survey," *Autonomous Robots*, vol. 22, no. 2, pp. 101–132, 2007.

[8] C. Lee, and Y. S. Xu, "Message-based evaluation in scheme for high-level robot control," *J. Intell. Robot. Syst.*, vol. 25, no. 2, pp. 109–119, 1999.

[9] X. M. Li, C. J.Yang, Y. Chen, and X. D. Hu, "Hybrid event based control architecture for tele-robotic systems controlled through internet," *J. Zhej. Univ. Sci*., vol. 5, no. 3, pp. 296–302, 2004.

[10] V. M. F. Santos, and F. M. T. Silva, "Design and low-level control of a humanoid robot using a distributed architecture approach," *J. Vibr. Contr.*, vol. 12, no. 12, pp. 1431–1456, 2006.

[11] M. Liu, Z. L. Shao, M. Wang, H. X. Wei, and T. M.Wang, "Implementing hybrid operating systems with two-level hardware interrupts," *in Proc 28th IEEE Int. Real-Time Systems Symposium*, pp. 244–253, 2007.

[12] S. S. Srinivasa, D. Ferguson, C. J. Helfrich, et al., "HERB: a home exploring robotic butler," *Auton. Robot.*, vol. 28, no. 1, pp. 5–20, 2010.

[13] M. Liu, D. Liu, Y. Wang, M. Wang, Z. L. Shao, "On improving real-time interrupt latencies of hybrid operating systems with two-level hardware interrupts," *IEEE Trans. Comput.*, vol. 60, no. 7, pp. 978–991, 2011.

[14] http://www.ros.org/.

[15] J. M. Romano, J. P. Brindza, and K. J. Kuchenbecker, "ROS open-source audio recognizer: ROAR environmental sound detection tools for robot programming," *Auton. Robot.,* vol. 34, no. 3, pp. 207–215, 2013.

[16] B. Hannaford, J. Rosen, D. W. Friedman, et al., "Raven-II: an open platform for surgical robotics research. *IEEE Trans. Biomed. Eng.*, vol. 60, no. 4, pp. 954–959, 2013.

[17] P. Bouchier, Embedded ROS [ros topics]. *IEEE Robot. Autom. Mag.*, vol.18, no. 3, pp. 17-19, 2013

[18] Q. Yu, H. Wei, M. Liu and etc. A novel multi-OS architecture for robot application. *in Proc. of the ROBIO*, pp. 2301-2306, 2013.

[19] http://www.nuttx.org/.

[20] S. Cousins, "Exponential Growth of ROS," *IEEE Robot. Autom. Mag.*, vol. 18, no. 1, pp. 19–20, 2011.

[21] S. Cousins, B. Gerkey, K. Conley, and W. Garage, "Sharing Software with ROS," *IEEE Robot. Autom. Mag.*, vol. 17, no. 2, pp. 12–14, 2010.

[22] S. Cousins, "ROS on the PR2," *IEEE Robot. Autom. Mag.*, vol. 13, no. 3, pp. 23–25, 2010.