22

Networking Multiple Robots for Cooperative Manipulation

M. Moallem

1. Introduction

In this chapter, the development of an open architecture multi-robot system is studied. The environment consists of five serial-link robot manipulators operated using embedded control computers. The robot control computers are connected together through a network of supervisory computers. A preemptive multi-tasking Real Time Operating System (RTOS) running on the supervisory computers is used to perform supervisory and cooperative tasks involving multiple robots. The software environment allows for controlling the motion of one or more robots and their interaction with other devices. Development of modular components is discussed in this chapter along with typical laboratory procedures. The environment can be used to develop software for various robotic applications such as scheduling robotic tasks, cooperative manipulation, collision avoidance, internet-based telerobotics, and other networked robotic applications.

2. Overview of Networked Multi-robot Systems

With the advent of new computing, sensor, and actuator technologies, the application of robotic systems has been growing rapidly in the past decade. Robotic systems were originally developed due to their capability to increase productivity and operate in hazardous environments. In recent years, robotics has found its way to a completely new range of real-world applications such as training, manufacturing, surgery, and health care (Bernard et al., 1999; Craig, 1997; Goldberg et al., 2000; Taylor and Stoianovici, 2003). From the advanced manufacturing domain to daily life applications, Internet-based telerobotic systems have the potential to provide significant benefits in terms of telepresence, wider reachability, cost effectiveness and maximal resource utilization. Challenging problems with regard to Internet-based telerobotic systems include such issues as uncertain Internet time delays (Luo and Chen, 2000), system reliability, interaction capability (Schulz al., 2000), and augmented Human-Machine interfaces. Due to the emergence of new areas in the field of

Source: Industrial-Robotics-Theory-Modelling-Control, ISBN 3-86611-285-8, pp. 964, ARS/plV, Germany, December 2006, Edited by: Sam Cubero

robotics, there is a growing need for applications that go beyond classical ones such as simple pick-and-place operations involving a single robot.
Many conventional robot manipulators are *closed architecture,* meaning that the user does not have direct access robot's sensory data and actuator inputs. To operate a robot, the user is usually confined to a *Robot Programming Language* (RPL) that is specific to the robotic system being used (Craig, 1997). This is restrictive in many cases, including the robotic applications requiring coordination of such robots. For example, in developing robotic work-cells that require interaction of two or more robots at a time, there is a growing need for robots to share and exchange information through a network. Use of an RPL is restrictive in such cases due to the limited capability of the programming environment. In this work we present a laboratory setup that can be utilized in order to perform tasks requiring multi-robot scheduling and cooperation tasks using industry grade manipulators. The objective is to create a flexible software environment so that the robot programmer can perform robotic tasks using a programming language such as C/C++ and a real-time operating system.

3. Interconnection of Multiple Robotic Systems

In this section an overview of a multiple robotic system is presented. The setup consists of stand-alone robotic systems which are interconnected through a computer network to be used in cooperative applications.

3.1 Hardware and Software Configuration

Figure 1 illustrates a multiple robotic system comprised of three 6 degree-of-freedom (DoF) and two 7-DoF robots, all from *CRS, Inc.*, located at the Robotics Laboratory, University of Western Ontario, Canada. The two 7-DoF robots are mounted on movable tracks while the other three 6-DoF robots are mounted on stationary bases. Each robot is equipped with a gripper controlled by a servo motor and a 6-dof force/torque sensor. The computing environment is comprised of a host-target architecture. The target machines consist of Pentium computers running under the *VxWorks* real-time operating system from WindRiver Systems, Inc (www.wrs.com). The host machines are used for system monitoring and development tasks and run under the Solaris or Microsoft operating systems.

3.2 Networking and Communication Configuration

The local networking and communication platform utilizes two types of mechanisms as shown in Figure 2, consisting of *serial port* which connect target

machines to robot controllers, and an *Ethernet network*, which links together all the target and host machines. Many commercial robots use serial communication between the host computers and the robot controllers. In this setup, we use the RS232 serial lines to transmit control commands and sensory information between the target machines and robot controllers. The robot motion commands are issued by a target machine and are sent to the robot controllers. The robot controllers also transmit sensory information such as gripper distances, force sensor readings, and the status of executed commands, back to the target machines. Similarly, the target machines transmit sensory and command data through the network to other machines. The robot controllers are embedded computer systems without network connectivity and standard operating system support. Lack of network connectivity is a main drawback of many conventional robot controllers. In Figure 2, the three target machines run under the VxWorks real-time operating system. However, any other operating system that supports networking tools and inter-task synchronization and communication primitives such as semaphores, message queues, and signals, can be used to do the same job.

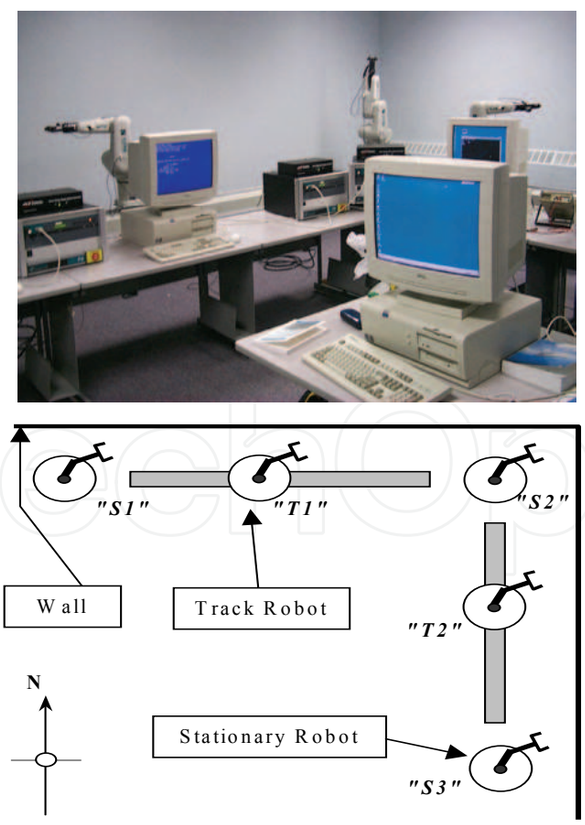


Figure 1. Multi-robot system (left) and layout of the robots (right)

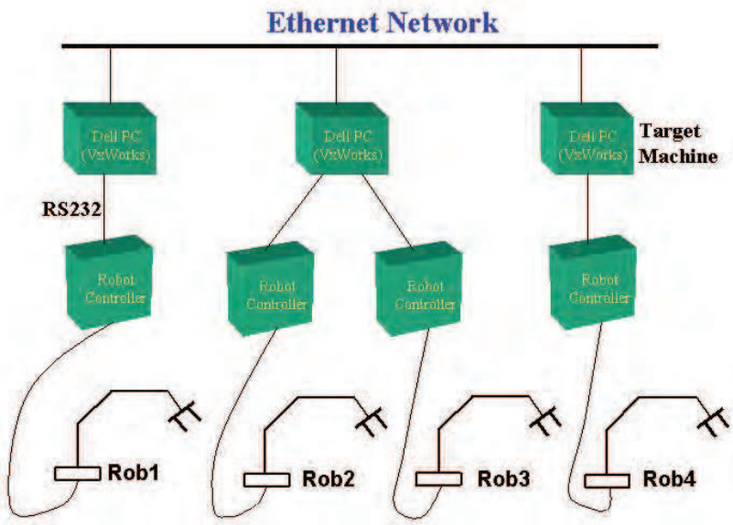


Figure 2. Configuration of the networked robotic system shown in Figure 1

3.3 Use of Real-time Operating Systems

Good practice in software engineering encourages the use of object-oriented programming for developing application software (Pressman, 1997). The main aspects of object-orientated programming are encapsulation, inheritance, modifiability, and reusability. In this regard, robotic systems are no exception. It is desirable to have software modules that can be easily ported to other platforms, to be modifiable, and can be reused for different robotic applications. This is particularly desirable in a laboratory setup where the functional and non-functional requirements of projects can change. Therefore, the availability of certain software modules would make it convenient to develop or modify code for new applications. On the other hand, the computer technology has got so powerful that an operating system can be used to develop and run applications on embedded computers. Nowadays operating systems are found in many devices and systems such as cell phones, wireless access points, robotics, manufacturing, and control applications. Many applications, including robotics, are real-time meaning that the computer must not only perform the calculations and logical operations correctly, but it must also perform them on time. In other words, correctness is as important as timeliness. Moreover, complex operations require modular programming which can be facilitated by using a real-time operating system. The operating system is responsible for operations such as controlling and allocating memory, prioritizing the execution of tasks, controlling input and output devices, networking operations, and managing files. The software developed using operating system facilities can be changed or modified easily without having to scrap the whole program.

4. Application Development for Distributed Robotic Applications

Distributed networked systems are increasingly becoming popular in industry, education, and research (Hung, 2000). Networked systems have the advantage of greater flexibility and better distribution of computing resources when compared to stand alone systems. Different networking architectures and protocols have been used in automation and control systems such as DeviceNet (DeviceNet Vendors Association, 1997), ProfiBus (http://www.profibus.com/), Manufacturing and Automation Protocol (Raji, 1994), ControlNet (ControlNet International, 1988), and Ethernet (see for example, Tanenbaum, 1996). Evaluation of the performance of these networks has been reported in the literature, for example in (Lian, *et al*., 2001) and (Hung, 2000). The emergence of *networked systems* on the factory floor is driving the automation industry to embrace new network technologies. For improved performance and cost efficiency, robots used on a factory floor should be enabled to provide data related to manufacturing and process operations to the management in real-time and preferably using non-proprietary networks. In the following, an outline of the software framework for supervisory control of the robots depicted in Figures 1 and 2 is presented.

4.1 Application Development under a Real-Time Operating System

Real-time operating systems have emerged in the past decade to become one of the basic building blocks of embedded computer systems, including complex robotic systems. A modular approach to software development for time critical embedded systems calls for decomposition of applications into multiple tasks and use of operating system primitives. A real-time operating system can be used to run on the supervisory computers such as the Pentium computers shown in Figure 2. We have used the *Tornado* development environment which provides a graphical user interface and tools for developing real time multitasking applications under VxWorks (www.wrs.com). However, any other real-time operating system can be used for this purpose. In Figure 2, once the programs are compiled and linked, the tasks can be downloaded into the memory of the PC workstations running *VxWorks*. These computers are used as supervisory controllers that enable communication between robots through the network communication ports.

4.1.1 The Robot Module

The starting point for implementing modular software for robotic applications is representing the robot as a class consisting of private data attributes and member functions as shown in Figure 3.

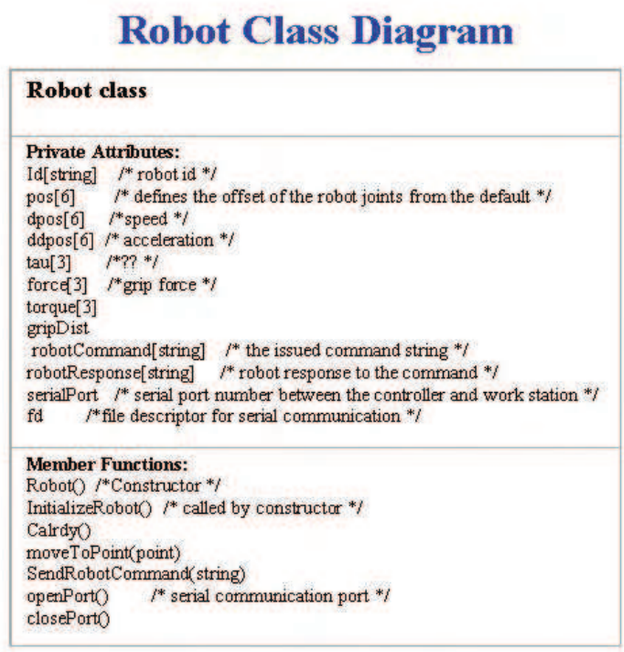


Figure 3. The *Robot Class* attributes and functions

In the class diagram of Figure 3, the robot identification, position, speed, torque, and other variables are defined as the attributes of the robot object. The commands sent to the robots are string variables stored in the robotCommand[ ] array. Similarly, the status received for each robot after executing a command is stored in the robotResponse[ ] array. The communication between the Pentium PCs in Figure 2 with the robot controller is two-way which is performed through the RS-232 serial interface. The serialPort attribute in Figure 3 is used to identify which serial port each robot object is using for communication. The member functions of the Robot Class shown in Figure 3 are used to initialize the robot object using *InitializeRobot()*, move the robot to a calibrated position using *Calrdy()*, move the robot to a particular point using *moveToPoint()*, send a command using *SendRobotCommand(),* and to open or close a serial port using *openPort()* and *closePort()*, respectively. If needed, the above class can be modified or other classes can be inherited from it.

One benefit of modular software development is the convenience of developing the modules one by one. After finishing each part, testing and debugging can be performed on other parts to be implemented. As a result, the final integration and testing can be done without much difficulty. In the following we discuss some of the projects that have been performed using this environment.

4.1.2 Controlling Robots through Serial Ports

Consider the implementation of a cooperative robotic task to be performed by the two robots indicated in Figure 4, in which the PC communicates with robot controllers through the RS-232 serial ports. Note that the tasks on the PC workstation are running concurrently using a real-time operating system (VxWorks in this case).

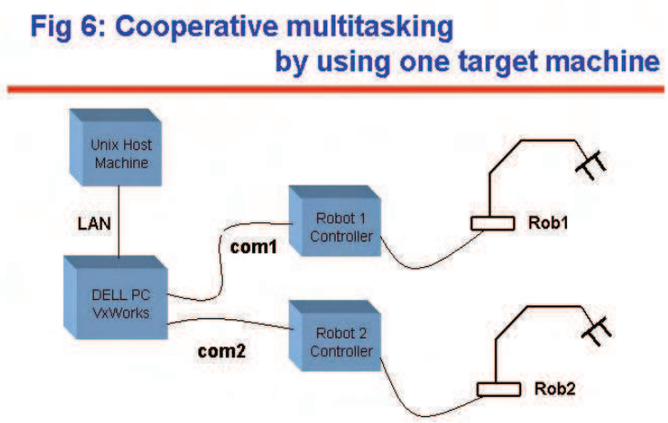


Figure 4. Cooperative multitasking by using one VxWorks station

Before running the system under *VxWorks*, the robot programming language *robcom* is used to send commands to the robot from the application shell, which is similar to MSDOS or UNIX prompts. For example, by issuing the command "*joint 5, 20*", the robot's fifth joint will rotate 20 degrees clockwise. This allows for the commands sent to the robot to be tested at the command prompt level before executing them from a program. The second step involves sending the commands through the serial ports using a high level program running under *VxWorks* instead of the application shell. For example, the function interface *SendRobotCommand()* in Figure 3 is written to send commands through the serial port, or the *moveToPoint()* command is to move the robot to a previously taught positions in the robot's workspace.

4.1.3 Object Handling

In this demonstration, one robot catches an object and passes it to a second robot. The goal is to develop code for coordination of motion of two robots using synchronization mechanisms such as *semaphores* provided by the operating

system. A semaphore is a token which if available, will cause the task to continue and if not available, will block the calling task until the token becomes available. At the beginning of the program, two robot objects are declared and initialized. The first robot is programmed to move and fetch the object from a known pick-up point that has been previously taught to it. Meanwhile, the second robot moves to the delivery position and waits for the first robot to deliver the object. A waiting mechanism is implemented using an empty semaphore by issuing a *semTake()* command in VxWorks which causes the task to block until the semaphore token becomes available. When the first robot has reached the delivery point and is ready to deliver the object, it releases the empty semaphore. The task running the first robot then unblocks, opens its gripper, and the process of transferring the object is completed. When the robot catches the object, it moves toward the release point where it allows the other robot to move to its final destination. At the end, both robots move to their calibration positions.

4.1.4 Network-based Cooperative Control of two Robots

Network communication allows more than two robots to perform cooperative tasks concurrently. In this scenario, socket programming under TCP/IP is used for communication between the *VxWorks* workstations in a client-server configuration. A server socket is assigned a well-known address which is constantly listening for client messages to arrive. A client process sends messages to the server via the server socket's advertised address. The hardware setup for this experiment is shown in Figure 5.

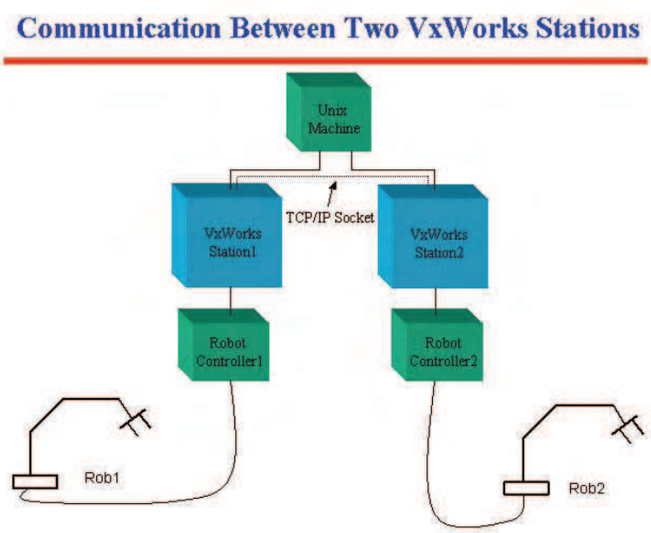


Figure 5. Hardware setup for TCP/IP communication

As indicated in Figure 5, two robots are connected to two different *VxWorks* workstations. The client program is run on one VxWorks workstation and the server program is run on the other station simultaneously. The previous demonstrations can be performed on this configuration too. For example, one robot can catch an object and pass it to a second robot in a similar manner as discussed before but by using the network interface. In this case, a set of special strings containing robot commands are defined on both the client and the server sides. These strings are then transmitted through sockets to perform cooperative tasks. The client-server mechanism can be used for synchronizing tasks with each other. At the beginning of the program, two robot objects are declared and initialized. In the server routine, a TCP socket is initialized, which listens for connection from the client while performing its own task. In the client routine, a socket is initialized and the connection request is sent to the server. After the connection is established, both client and the server can synchronize their operations.

4.1.5 Robot Interaction with Input-Output Modules

There are many situations that robots must coordinate their operations with external devices. For example in a smart manufacturing workcell, robots have to grab parts from devices such as indexers, or deliver parts to them in a synchronized manner. A rotary indexing table is a simple example that simulates part of a manufacturing process. The indexing table can rotate by means of a stepping motor. The motor controller is interfaced with a *VxWorks* station and a digital I/O card. The indexer and robot can be connected to the same computer or to separate computers on a shared network. The situation is similar to the previous example in part E. The program that controls the indexing table operation is spawned as a separate task which can coordinate its operation with other robots, for example by using semaphores or a TCP/IP client-server mechanism on a network.

4.1.6 Other Multi-Robot Operations

Several projects related to scheduling and cooperative operation of robots similar to those used in a manufacturing work-cell have been carried out using the setup described in this chapter. For example, referring to Figure 1, an object handling operation was developed where the first robot takes an object from a person and delivers it to the second robot. Then, the second robot delivers the object to the third robot, and so on, until the object is delivered to the fifth robot.

Another project was related to visualization of a robotic work-cell using the Matlab Virtual Reality Modeling Language (VRML) toolbox, from MathWorks, Inc. In this project, sensory data such as joint displacements are sent through

the network to a host computer that may be located in a control room. A visualization program written in VRML is running on the host computer. This computer obtains real-time sensory data from supervisory robot computers on the network and presents to the operator a visualization of the robotic work-cell. A main advantage of using this scheme over sending data through a camera vision system is the small bandwidth required for sending sensory data, as opposed to a relatively large bandwidth required for transmitting picture frames when a vision system is used.

The environment has also been used in a distributed robotic system with Internet connectivity where robot operations can be monitored and operated from the Internet (Wang *et al.,* 2003). The scheduling of robotic work-cells used in manufacturing has been another topic, where issues such as checking for deadlock situations, or scheduling the operation of multiple robotic systems with different timing requirements have been addressed (Yuan *et al.,* 2004).

5. Conclusion

In this chapter, some aspects of developing modular software for controlling robot operations were discussed. Many commercial robots have a closed architecture, which makes them difficult to program for certain applications involving multiple robots. A networked robotic system offers interesting possibilities in terms of developing novel applications. With the recent advancements made in the networking technologies it is important that students and engineers taking courses or projects in robotics and automation be familiar with the capabilities offered by new technologies.

6. References

Bernard, C., Kang, H., Sigh, S.K. and Wen, J.T. (1999), "Robotic system for collaborative control in minimally invasive surgery", *Industrial Robot: An international Journal*, Vol. 26, No. 6, pp. 476-484.

Craig, C.G. (1989), *Introduction to Robotics: Mechanics and Control*, Addison-Wesley, Boston, MA.

Goldberg, K., Gentner, S., Sutter, C. and Wiegley, J. (2000), "The mercury project: A feasibility study for Internet robots" *IEEE Robotics & Auto. Magazine*, Vol. 7, No.1, pp. 35-40.

Pressman, R. (1997), *Software Engineering: A Practitioner's Approach*, McGraw-Hill, New York, NY.

DeviceNet Vendors Association (1997), *DeviceNet Specifications*, 2nd. ed. Boca Raton, FL.

Raji, R.S. (1994), "Smart Networks for Control," *IEEE Spectrum*, Vol. 31, pp. 49-55, June 1994.

ControlNet International (1988), *ControlNet Specifications*, 2nd ed. Boca Raton, FL.

Tanenbaum, A.S. (1996), *Computer Networks*, 3rd ed. Upper Saddle River, Prentice Hall, NJ.

Lian, F.-L. Moyne, J.R.; Tilbury, D.M. (2001), "Performance evaluation of control networks: Ethernet, ControlNet, and DeviceNet," *IEEE Control Systems Magazine*, Vol. 25, No. 1, pp. 66-83, 2001.

Hung, S.H., (2000), "Experimental performance evaluation of Profibus-FMS," *IEEE Robotics & Automation Magazine*, Vol. 7, No. 4, pp. 64-72.

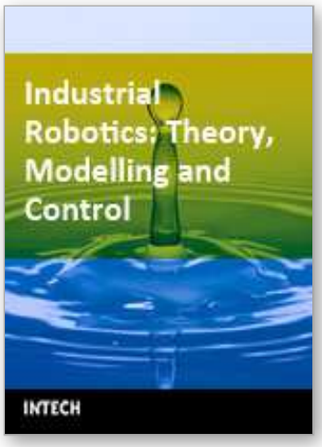
Taylor, R.H. and Stoianovici, D. (2003), "Medical Robotics in Computer Integrated Surgery," *IEEE Transactions on Robotics and Automation*, vol. 19, pp. 765–781.

Luo, R.C., and Chen, T.M. (2000), "Development of a Multibehavior-based Mobile Robot for Remote Supervisory Control through the Internet", *IEEE/ASME Trans. on Mechatronics,* Vol.5, No.4, pp. 376-385.

Schulz, D., Burgard, W., Fox, D., Thrun, S. and Cremers, A.B., (2000), "Web Interfaces for Mobile Robots in Public Places", *IEEE Robotics & Auto. Magazine*, Vol. 7, No.1, pp. 48-56.

Wang, X-G., Moallem, M. and Patel, R.V., (2003), "An Internet-Based Distributed Multiple-Telerobot System," *IEEE Transactions on Systems, Man, and Cybernetics, Part A,* Vol. 33, No. 5, pp. 627- 634.

Yuan, P., Moallem, M. and Patel, R.V. (2004), "A Real-Time Task-Oriented Scheduling Algorithm for Distributed Multi-Robot System," *IEEE International Conference on Robotics and Automation*, New Orleans, LA.



Industrial Robotics: Theory, Modelling and Control

Edited by Sam Cubero

ISBN 3-86611-285-8

Hard cover, 964 pages

Publisher Pro Literatur Verlag, Germany / ARS, Austria

Published online 01, December, 2006

Published in print edition December, 2006

This book covers a wide range of topics relating to advanced industrial robotics, sensors and automation technologies. Although being highly technical and complex in nature, the papers presented in this book represent some of the latest cutting edge technologies and advancements in industrial robotics technology. This book covers topics such as networking, properties of manipulators, forward and inverse robot arm kinematics, motion path-planning, machine vision and many other practical topics too numerous to list here. The authors and editor of this book wish to inspire people, especially young ones, to get involved with robotic and mechatronic engineering technology and to develop new and exciting practical applications, perhaps using the ideas and concepts presented herein.

IntechOpen