QRobot – A Multitasking PC Based Robot Control System

N. Costescu, M. Loffler, E. Zergeroglu and D. Dawson

Department of Electrical and Computer Engineering
Clemson University / 105 Riggs Hall
Clemson, SC 29634-0915
Fax: (864) 656-7220
Tel: (864) 656-7708

ncostes@eng.clemson.edu
loffler@eng.clemson.edu
ezerger@eng.clemson.edu
ddawson@eng.clemson.edu
http://ece.clemson.edu/crb

Submitted to the
Conference on Control Applications
Trieste, Italy
September 1998

Keywords: Real-time, Control, Robotics, PC, QNX, Mechatronics, RCCL, ARCL

Abstract

In this paper, we describe QRobot, a low level real-time PC-based system for controlling PUMA robot manipulators. Specifically, we illustrate how a standard PUMA controller can be retrofitted with interface boards that allow it to be connected to an industry standard PC. All computational units are removed from the existing PUMA controller, and the PC assumes the role of computing the control strategy. A low level control is implemented for a PUMA 560 manipulator. The low level control software is then connected to two high level software environments (RCCL and ARCL) for purposes of high level trajectory generation, kinematics, inverse kinematics, *etc*. Excellent tracking results are obtained with both systems. Some limitations of the two high level software environments were discovered during the development of QRobot. These limitations provide the motivation for the future development of a new object oriented robot control system.

1 Introduction

Over the last ten years, the U.S. Department of Energy has become increasingly concerned with the decontamination and decommissioning (D&D) of numerous laboratory/production facilities that contain hazardous materials. Since the use of telerobotic systems provides a means for achieving human performance capabilities without human exposure, many researchers have proposed the use of advanced sensor-based telerobotic systems to accelerate ongoing D&D operations and overcome the potential safety problems. Indeed, there are probably few problem domains that are more appropriate candidates for telerobotic technology than D&D operations. To facilitate the development of low cost teleoperated systems, we are currently investigating the feasibility of designing a PC-based robot control system for PUMA robot manipulators as part of a Semi-Autonomous Decommissioning of Hazardous Waste Project for the U.S. Department of Energy [1].

A standard PUMA robot manipulator is controlled by a VAL-II based Unimation Mark II controller. There are several disadvantages or limitations of the VAL-II based Mark II controller system, many of which are attributable to the age of the product. To overcome some of the disadvantages of the VAL-II based Mark II controller system, RCCL/RCI (Robot Control C Library/Real Time Control Interface) [2] was developed; however, RCCL/RCI does not allow for modification of the low level control strategy or the incorporation of additional sensor data into the low level control system. To overcome some of the limitations of RCCL/RCI, we propose the development of a flexible PC-based PUMA robot manipulator control environment that would allow the user to change the low level control strategies and use a wide variety of sensors in the trajectory generator and low level control.

The combination of the following three technical developments has made the PC-based PUMA retrofit possible: i) the advent of high-speed PC CPUs, ii) the availability of a hardware interface for the existing Mark II controller, and iii) the availability of hard real-time operating systems for PCs. The proposed PC-based control system involves a hardware retrofit of the Mark II controller, low level control software development, and high level application program interface and trajectory generation software development.

2 Mark II / Val-II System

A standard PUMA Robot Manipulator is controlled by a VAL-II based Unimation Mark II controller. Basic knowledge of the Unimation controller architecture is necessary to understand the disadvantages and shortcomings of this aging architecture (for more details on the controller see [3]). To provide the necessary background on the Unimation controller, Table 1 lists the major components of the system, and Figure 1 depicts a block diagram of the Mark II control system.

1	DEC LSI-11/2 computer with ADAC parallel interface board, DLV11-J serial interface board, CMOS board, and EPROM board
2	Servo interface board
3	Six 6503 based digital servo boards
4	Two power amplifier assemblies
5	Power amplifier control board
6	Clock/terminator board
7	Input/output interface board
8	Two power supplies
9	High power function board
10	Arm cable board

Table 1 – Major Components of the Mark II Control System

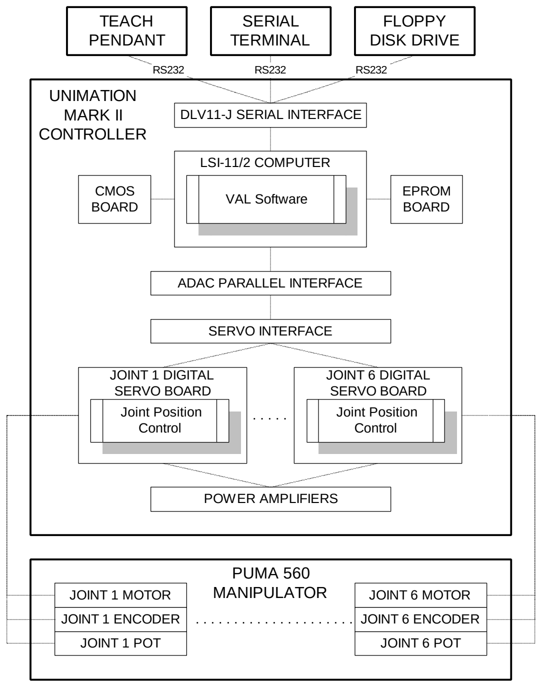


Figure 1 – VAL-II Based Mark II Control System Block Diagram

In this paper, we will primarily focus on the disadvantages and drawbacks with regard to components 1-3 in Table 1; hence, a brief description of these components and some related subsystems is given below.

LSI-11/2 Computer: The DEC LSI-11/2 computer is based on a 16-bit processor with up to 32K words of memory. This 1970's era processor [4] is used to compute the setpoints which make up the desired robot joint level trajectory. This processor communicates via the DLV11-J serial interface card to a floppy disk drive, a teach pendant, and a serial terminal. Commands are usually entered at the terminal or the teach pendant. The setpoint can be updated by the LSI-11/2 based software at one of several user selectable periods. The default update period is 28ms. Each update period new setpoints for each joint are transferred from the LSI-11/2 to the servo interface board, which in turn distributes the appropriate setpoint to each digital servo board.

Digital Servo Boards: There are six digital servo boards, one per joint of the robot manipulator. Each servo board consists of a 6503 8-bit 1MHz microprocessor with 2048 bytes of EPROM and 128 bytes of RAM, a counter/timer, and some parallel input/output capabilities [5]. Each processor implements the position loop for its joint at about 1KHz. Computations are performed using fixed-point triple precision (24 bit) arithmetic. A discussion of the low level control loop is immaterial to this paper and can be found in [5]. It is sufficient to say that it is a fairly straightforward computation which can be performed by the 6503 in approximately 924μs.

Peripherals: Connected to the Mark II controller are a floppy disk drive, a teach pendant, and a serial terminal. Programs entered from the terminal can be stored on the floppy drive. The teach pendant is used to move the robot manipulator directly either in Cartesian or joint space, and to program motions by moving the robot manipulator as opposed to entering desired end-effector configurations from the terminal.

Mechanical System: The final and most important piece of hardware is the PUMA 560 robot manipulator. This robot is a six degree of freedom revolute manipulator with a 2.2 pound static payload capability. PUMA robot manipulators are popular in industrial applications due to their accuracy, repeatability, and reliability. These robots are popular in academic settings because previously owned units can be obtained at reasonable prices.

Software: The VAL-II software for the Mark II controller allows the user to program robot motions using an English-like syntax. Once a VAL-II command is issued from the terminal, the LSI-11/2 computer computes the joint level trajectory and sends setpoints to the digital servo boards every 28ms until the motion is complete. The VAL-II software then notifies the terminal that a new command may be entered.

Disadvantages: There are several disadvantages or limitations of the VAL-II based Mark II controller system. The obsolete LSI-11/2 processor based computer with its limited computational abilities and memory does not permit a sophisticated trajectory generator to be implemented. The digital servo boards with their even more limited 8-bit 6503 processors and 128 bytes of RAM limit the complexity of the low level control that can be implemented. The 1 KHz sampling frequency is also fairly low by modern standards. Entering programs from a serial terminal and storing them to a low capacity floppy drive is a cumbersome method of software development, to say the least. Perhaps the greatest disadvantage is the fact that this system does not allow integration of arbitrary sensors into the trajectory generator or low level control loop. For example, with a VAL-II based system it is not feasible to implement a visual-based servo control architecture.

3 Overview of RCCL System

RCCL/RCI (Robot Control C Library/Real Time Control Interface) overcomes some of the disadvantages of the VAL-II based Mark II controller system for PUMA Robot Manipulators. An RCCL upgrade to a VAL-II based system consists of both hardware/software upgrades and retrofits and requires the purchase of some new hardware (Figure 2 depicts a block diagram of the RCCL based system).

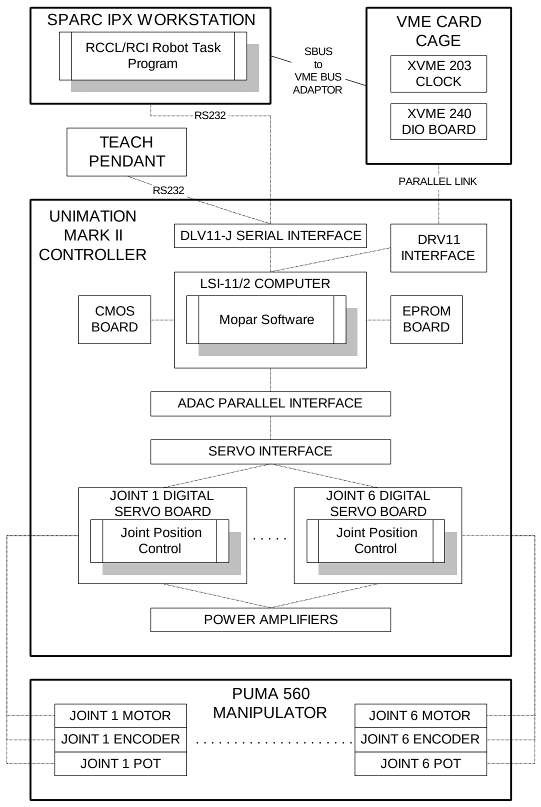


Figure 2 - RCCL Based Mark II Control System

RCCL Retrofit: As illustrated by Figure 2, the serial terminal is now replaced by a UNIX workstation. At the Clemson Mechatronics lab [6], a Sparc IPX workstation running SunOS is connected to a VME card cage via an SBUS to VME bus adapter. The VME card cage contains an XVME-203 real time clock board for generating real time interrupts, and an XVME-240 DIO board for performing parallel I/O to the Mark II controller during control program execution. An RS232 port on the Sparc workstation is connected to the Mark II console port for downloading and initialization of the new LSI-11/2 software (*mopar*) which replaces VAL-II. The Mark II controller is retained unaltered except for the addition of a DRV-11 parallel interface card which connects to the XVME-240 parallel board. The PUMA 560 robot manipulator remains unchanged. The replacement software *mopar* is a LSI-11/2 program which takes care of communicating with the digital servo cards, teach pendant, *etc*. *Mopar* is aware of RCCL/RCI, and must be downloaded via the serial link from the UNIX workstation each time the Mark II controller is powered on. The above RCCL/RCI retrofit was used successfully at the Clemson Mechatronics Lab for several projects, including [7], [8], and [9].

Advantages of RCCL: RCCL/RCI provides a rich library of functions for controlling virtually any type of robot. The system is extensible to multiple user-defined robots. RCCL programs are considered planning tasks while RCI programs are real time control level tasks. Sensor input may be integrated into the trajectory generator at the RCI level. For example, an impedance based control for optimizing remote saw cutting was implemented using RCI level integration of a force sensor [9].

RCCL also provides a three-dimensional graphic robot simulator; moreover, the same RCCL program that is developed to move a real robot can first be run with the simulator. This feature is extremely useful for debugging robot tasks that might otherwise result in collisions between the robot and itself or its workspace. The simulator also allows the new RCCL user to experiment with the system without fear of damaging the manipulator.

By utilizing programs written at the RCI level, the user can generate a dynamic desired trajectory in real time. The generation of desired trajectories in real time was not possible with the old VAL-II based system. In addition, because the workstation running RCCL/RCI is far more powerful than the LSI-11/2 computer, more complex trajectories can be implemented (i.e., minimization of kinetic energy, singularity avoidance, *etc*. [10]).

Disadvantages of RCCL: Though RCCL/RCI addresses many of the shortcomings of the original VAL-II based control system and provides a robust environment for developing robot task programs, there are some issues that RCCL/RCI does not address. One is the inability to implement a user defined low level control strategy. Another is the inability to incorporate additional sensor data into the low level control loop. These shortcomings are directly related to the fact that an RCCL based system still uses the Mark II controller's original computational hardware (the digital servo boards) to implement the low level control.

The RCCL/RCI retrofit also consists of several complex procedures. For example, the retrofit (on our Sparc) involved installing an SBUS to VME bus adapter and configuring the SunOS kernel to use the bus adapter and the XYCOM boards installed in the VME card cage. The SunOS kernel had to be patched to implement the real-time capabilities of RCI; moreover, some system devices had to be created [11]. This installation required some UNIX system administration, system programming, and hardware interfacing skills. Despite the initial complexity, the system has performed well, except for occasional SunOS kernel panics that are probably due to insufficient real memory (our Sparc is equipped with 32MB RAM). These kernel panics would be devastating in a production situation because the control system simply stops. Since the user level program that is running on the Sparc workstation has crashed, the PUMA's electromagnetic brakes engage to prevent the manipulator from being damaged, but the task the robot is performing is interrupted with no means of recovery.

An additional disadvantage is cost. While the RCCL/RCI software is available free of charge, the computer hardware necessary to implement the RCCL/RCI retrofit is expensive. A Sparc workstation (or any of the other UNIX workstations supported) is quite costly when compared to a PC system. A VME card cage and bus adapter, as well as the XYCOM boards must also be purchased. Finally, the labor cost of installing and configuring the system can be quite high.

4 Motivation for a New PUMA Retrofit

There were three main motivating factors for development of a new PC-based PUMA robot manipulator control system: cost, complexity, and flexibility. The cost and complexity of an

RCCL based system were discussed in the previous section. Flexibility refers to the ability to implement arbitrary user-defined low level control strategies which can easily integrate sensor information into the low level control. Flexibility also refers to the ability to easily use a wide variety of sensors in the trajectory generator, low level control, and user programs. While an RCCL/RCI based simplified impedance control was implemented by using force sensor measurements to modify the desired position trajectory in [9], true force based control requires that force sensor measurements be injected into the low level controller. The ability to integrate additional sensors, such as ranging lasers, sonar, cameras, *etc*. into the low level control would also allow the user to implement more complex control strategies (e.g., vision based control). With the current digital servo boards in the Mark II, none of these more advanced operational modes are feasible.

Implementation Issues: A new PUMA retrofit that would allow user defined low level control strategies to be implemented must completely eliminate the LSI-11/2 and the digital servo boards and replace this hardware with components that have the appropriate interfacing and computational capabilities. In addition, the new PUMA retrofit should have the capability to provide a simple means of programming the low level control and adding additional sensors. Once a platform for implementing low level control programs is developed, the next step is to connect this low level controller to a high level trajectory generator. One option is to use an existing system such as RCCL or ARCL [12]. Another option is to develop a system from scratch. The first option (interfacing with RCCL and ARCL) is explored below while the second option is the direction of our future work.

5 PC-Based PUMA Retrofit

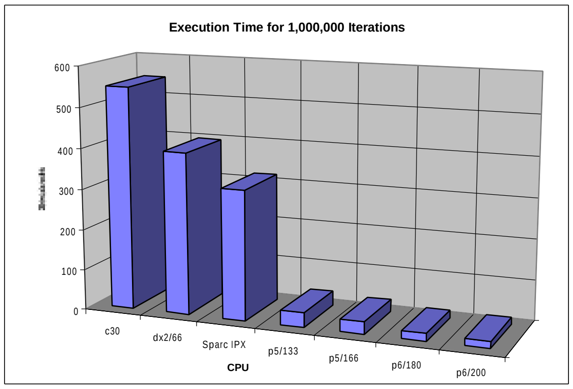
The combination of the following three technical developments has made the PC-based QRobot PUMA retrofit possible: i) the advent of high-speed, PC-based microcomputer CPUs, ii) the availability of a hardware interface that can connect an external controlling computer directly to the PUMA servo amplifiers and arm cable in the existing Mark II controller, and iii) the availability of hard real-time operating systems for PCs (*e.g.*, the QNX operating system [13], [14]). Specifically, the availability of cheap, powerful PCs and efficient PC real-time operating systems make the implementation of a PC-based robot control system possible; moreover, the TRC [15] board set for the Mark II controller makes a PC-based retrofit quite painless.

Real-Time PC-Based Control: Due to the economies of scale and increased CPU power, personal computers have become viable, cost-effective alternatives to workstations in engineering applications. An Intel Pentium processor easily outperforms a Sparc IPX while costing less than a thousand dollars. Personal computers also have a lower maintenance cost associated with them, both in terms of cost of parts, and in terms of expertise required to maintain them. PC-based systems are also inherently less complicated than UNIX workstations.

Table 2 below shows the results of running a control benchmark on various hardware platforms. The benchmark consists of executing an extremely complex nonlinear control program for a two degree of freedom revolute manipulator with flexible joints. All calls to perform A/D and D/A were replaced with stubs, and the time required to execute one million iterations was recorded. Speedup relative to a baseline Texas Instruments C30 DSP board based system was also computed. As the table shows, modern PC CPUs such as the Intel Pentium and Pentium Pro outperform the C30 and the Sparc IPX by at least a factor of 15 (an explanation of this phenomenon is found in [16]). Basically, the ratio of floating point instructions to integer and other instructions in a control program is very low. Consequently, the time required to execute general purpose instructions dominates the time required for floating point instructions, making a CPU's FLOPS rating much less significant than one might assume. Therefore, an Intel 486 DX2/66, which is rated at 3 MFLOPS, can outperform a C30, which is rated at 33 MFLOPS (*i.e.*, for the implementation of this particular control algorithm).

OS	CPU	Execution Time (sec)	Speedup
-	C30	548	1.00
Win 3.1	486DX2/66	398	1.38
SunOS	Sparc IPX	319	1.72
Win 3.1	P5/133	36	15.22
Win95	P5/166	30	18.27
NT 4.0	P6/180	19	28.84
NT 3.51	P6/200	17	32.24

Table 2 - Execution Time and Speedup



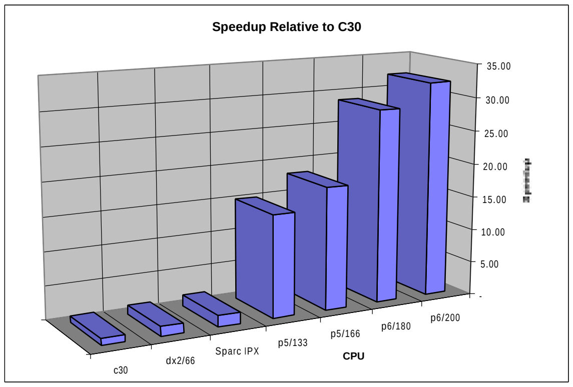


Figure 3 - Execution Time of 1,000,000 Iterations

Figure 4 - Speedup on Various CPUs

Because of the increased computing power and memory capacity of these systems, a PUMA retrofit can be designed that will allow the user to implement more complex low level control strategies. In addition, due to the wide range of sensors with ISA or PCI bus interfaces, there is greater flexibility in fusing sensor data into a low level control program (*i.e.*, interface libraries for ISA and PCI bus based video capture boards, laser ranging devices, *etc*. are widely available).

TRC Hardware Interface: The main hardware component of our new controller architecture is the TRC004 general purpose servo controller board. This board provides eight channels of buffered analog output, eight channels of analog input, six quadrature encoder inputs, 7 discrete output bits and 6 discrete input bits. In our application the TRC004 is connected to the PC ISA bus via a TRC006 board, and to the Mark II controller via the TRC041 PUMA cable card set. The TRC004 board is produced by Trident Robotics and Research Inc., a small company which manufactures interface boards for Unimation control systems. Although there are many motion control interface boards available now ([17], [18], *etc*.), the TRC boards are specifically designed for use with PUMA robot manipulators and controllers and provide an almost "plug and play" solution. That is, installation of the TRC004 in the Mark II only requires point to point soldering of connections to the Mark II backplane; moreover, the TRC041 Cable Card Set removes the need for soldering by providing an interface between the TRC004 and the Mark II Arm Bus. Connections to the motors, encoders, potentiometers, *etc*. on the PUMA robot are made through the Arm Bus slots rather than by point to point solder connections to the Mark II backplane. Although the use of the TRC041 card set is not absolutely necessary, its use eliminates a large amount of tedious soldering work and reduces the risk of damaging the backplane. The TRC006 (a bus decoder/buffer card for the PC ISA bus) provides the link between the PC and the Mark II controller. This hardware connection allows the PC to execute low level control programs while retaining the existing Mark II controller connections to the power amplifiers, joint actuators, and encoders. The TRC006 board uses 64 bytes of ISA bus I/O address space when connected to a TRC004. Encoder and potentiometer values are read from the TRC006, and desired joint actuator voltages are written to it

Hard Real-Time Control: One of the requirements for successful implementation of a high-speed low level control program is deterministic response. If the only program being executed on the computer is the control program, then deterministic response is not too difficult to achieve (see [19] for a single tasking PC implementation of a PUMA controller). For reasons of cost and simplicity, it would be desirable to have the user interface, the high level trajectory generator, and the low level control programs all executing on the same PC. Hence, a multitasking operating system that provides deterministic response and priority based scheduling is necessary. In addition, the operating system must have low overhead, allowing most CPU cycles to be expended by performing useful computation such as control calculations or real-time data plotting, rather than being consumed by OS overhead such as performing context switches. Finally, the operating system should facilitate the development of hardware/software interfaces. The real-time microkernel based operating system QNX, developed by QSSL [20], meets all of these requirements. The availability of a high quality, efficient real-time operating system, like QNX, makes the PC a viable platform for real-time control. The feasibility of developing a real-time PC-based control environment was presented in [16] and [21] which describe QMotor 2.0, a QNX based real-time control environment developed for the testing of motor control algorithms.

6 QRobot Retrofit

The QRobot retrofit involved a hardware retrofit with the TRC board set, low level software development (*i.e.*, the design of a real-time servo subsystem), and high level software development (*i.e.*, connection of the servo subsystem to high-level trajectory generators).

Hardware Retrofit: The first step was to remove the digital servo boards, LSI-11/2 computer, CMOS and EPROM boards, and the ADAC interface board from the Mark II chassis. The TRC041 cable card set was then installed in the Mark II controller, thereby providing a connection between the backplane of the Mark II and the TRC004 board. The TRC004 board was then bolted to the chassis of the Mark II and connected to the TRC041 cable card set. Some minor adjustments to the TRC004 were necessary. These adjustments required simple instruments found in most control labs. Instructions for performing these adjustments were included with the TRC004 board, and the board's designer [22] was very helpful during this phase. Finally, the TRC006 was installed in the QNX based PC that is used to perform the control computations. This hardware retrofit effectively replaced the obsolete 1970's era controller with a Pentium based computer that has orders of magnitude greater computing power and memory capacity.

Figure 5 illustrates how the computational engine has been moved from the Mark II controller to the PC. Notice how much simpler the retrofitted system is when compared to Figure 1 and Figure 2.

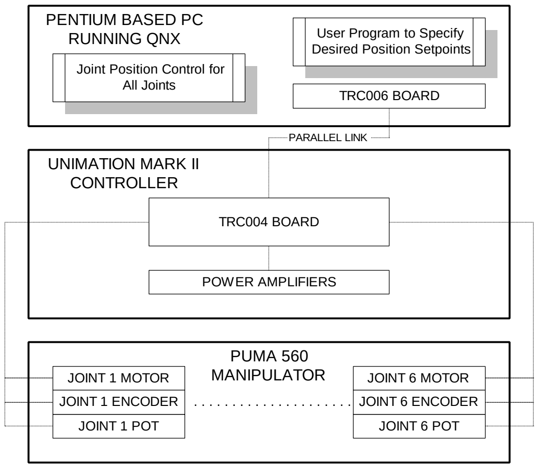


Figure 5 - QRobot System Without RCCL

Low Level Control (QRobot): In the original Mark II architecture, the digital servo boards contain the low level controller that ensures that link actuators move to the desired joint position. Each of the desired joint position setpoints are specified by a high level trajectory generator. Due to the disadvantages mentioned in sections 3 and 4, the digital servo boards must be replaced with a digital control program running on the QNX based PC. Since the original Mark II LSI-11/2 based trajectory generator produced new setpoints every 28ms, most high level replacements for this trajectory generator (RCCL, ARCL, *etc*.) have been written to produce new setpoints at the same rate (*i.e*., 28ms). QRobot was also designed to accept setpoints at this rate.

The low level control program is an implementation of a proportional derivative controller with static/coulomb friction compensation for all joints and gravity compensation for the second and third joints. The low level control program inputs the current joint position from the encoders and outputs the calculated control signal to the joint actuators. Joint velocities are manufactured via a backwards difference method. The manufactured joint velocity signals are then low pass filtered to eliminate excess noise. Figure 6 depicts the low level control structure.

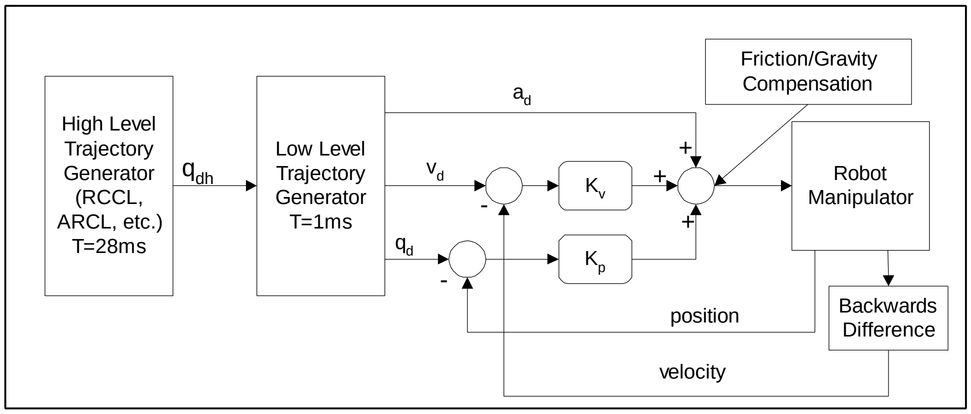

High Level Trajectory Generator (RCCL, ARCL, etc.) T=28ms
q_{dh}
Low Level Trajectory Generator T=1ms
a_d
v_d
q_d
K_v
K_p
Friction/Gravity Compensation
Robot Manipulator
Backwards Difference
position
velocity
+
-

Figure 6 – Low Level Control Structure

In Figure 6, the symbols a_d, v_d and q_d denote the desired acceleration, velocity, and position, respectively. The desired position setpoint generated by the high level trajectory generator (RCCL, ARCL, *etc*.) is denoted by q_{dh}, while K_p, K_v are position/velocity feedback gains, which are determined experimentally. The desired acceleration and desired velocity were calculated by taking the time derivative of the expression given for q_d.

To test the controller, a simple program that allowed the user to enter the desired joint setpoint position in degrees was written. To mimic a separate high level trajectory generator, a separate program was used to pass the q_{dh} vector to the low level control program using a QNX interprocess communication mechanism called message passing (in the final working system, the trajectory generator will be a separate process). The use of interprocess communication at this early stage made the integration of the low level control program with high level trajectory generators very easy. The low level control program was run at 1KHz, 2KHz, and 4KHz. Software QNX timers (for the 1 and 2 KHz runs) and a hardware timer (for all frequencies) were used to run the control with identical results (the software timers proved reliable). An RCCL program was written to move the PUMA from the RCCL park position to an end position at a constant speed and then back to the park position at triple that speed. Figure 7 shows the desired position trajectories for all joints while Figure 8 shows the position tracking error.

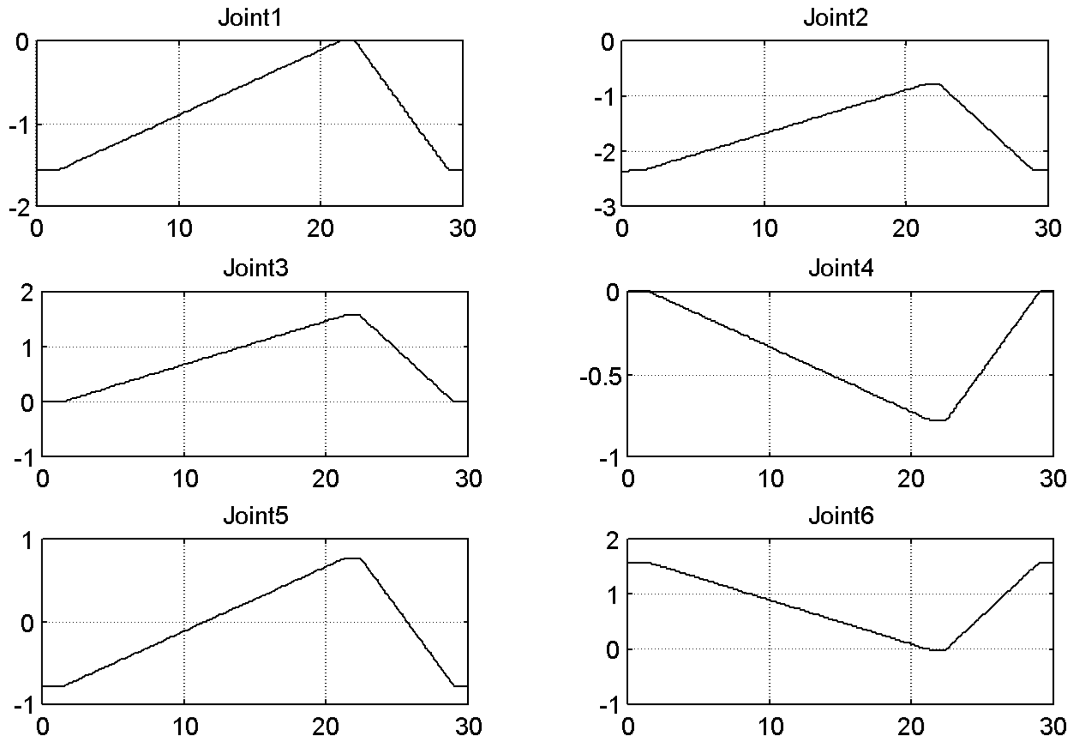


Figure 7 - Desired Joint Positions (Radians vs Seconds)

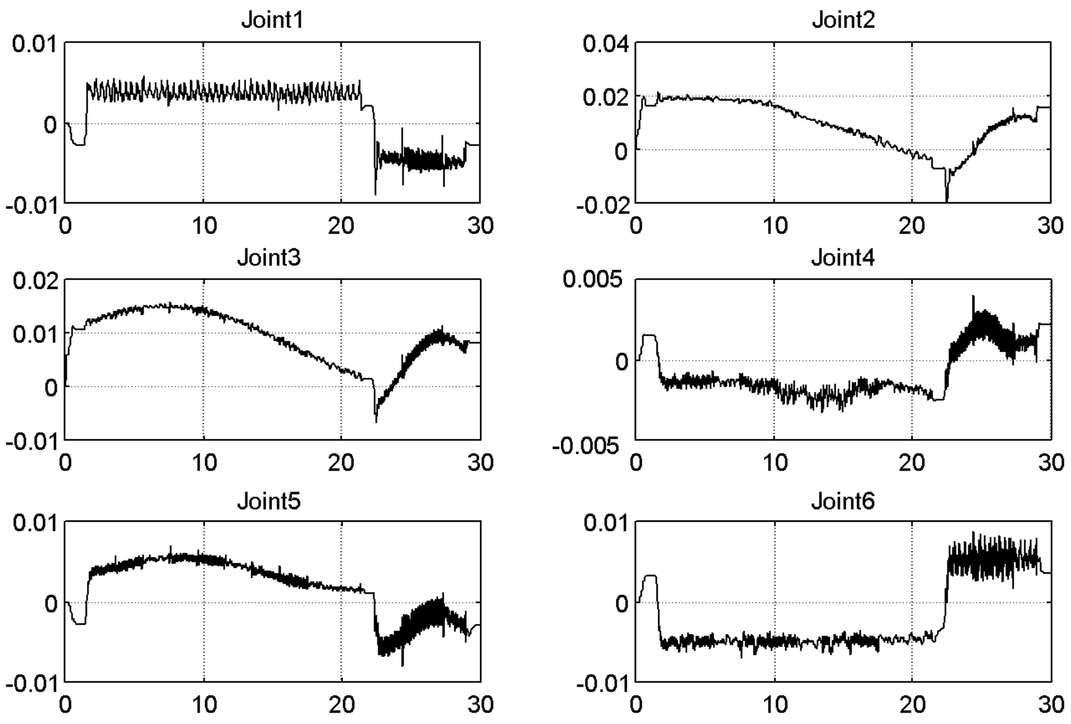


Figure 8 – Position Tracking Error (Radians vs Seconds)

High Level Software Development for RCCL Connection: The QRobot system provides a means for moving the robot to a desired setpoint position. In order to perform useful tasks, the robot must be able to follow trajectories specified programmatically (*i.e.*, an application program interface and a high level trajectory generator are needed). Since the RCCL high level trajectory generator has many desirable qualities, we developed an interface program to connect RCCL to QRobot. Figure 9 shows how the QRobot system was integrated with our RCCL workstation.

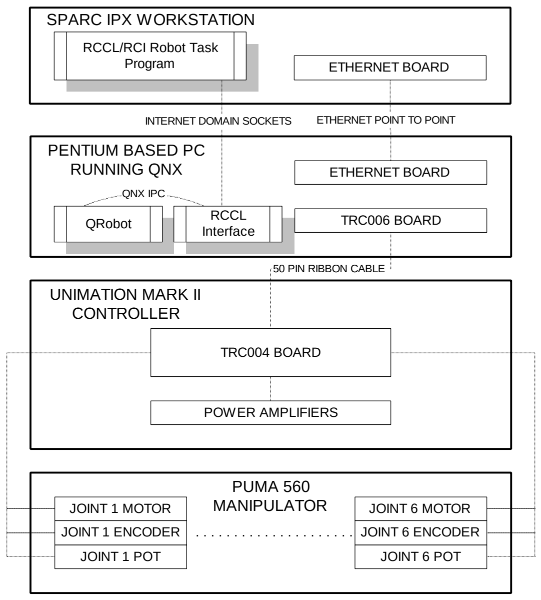


Figure 9 – RCCL/QRobot Architecture

Normally, when connecting RCCL to a robot controller, an RCCL I/O driver must be written and a communication protocol must be developed. In our existing RCCL system (see Figure 2), communication between the Sparc running RCCL and the Mark II controller is implemented via a parallel interface (with additional hardware such as an SBUS->VME bus adapter, *etc*.) However, for the QRobot retrofit, RCCL was connected to a QNX based PC which runs QRobot. Since both QNX and UNIX provide TCP/IP based Ethernet networking facilities, we used TCP/IP sockets to avoid having to write a communications protocol or a hardware device driver. In addition, since RCCL's 3-D simulator uses internet domain sockets to communicate with RCCL user programs, we decided to use the simulator service for QRobot connection rather than writing an RCCL I/O driver from scratch. The time savings in terms of software development effort is significant.

The communication between RCCL's trajectory generator and its 3-D robot simulator was implemented at two layers. The lower layer establishes a simple package oriented connection. This layer was implemented in the RCCL simulator device driver. While the trajectory generator is running, packages are transferred continuously in both directions over this connection at a specified rate. In the old RCCL system, packages were transferred between the RCCL program running on the UNIX workstation and *mopar* running on the Mark II's LSI-11/2 processor. For the PUMA 560, each data package contains the new desired setpoints for each joint along with other commands such as enabling or disabling arm power, *etc*. The packages received from the controller by the driver contain the current state of the robot (the current joint angles, information about the gripper, force/torque sensors, *etc*.)

In the existing RCCL system, the robot simulator interface performs both lower and higher layer communication on the other side of the connection. In the RCCL/QRobot system, the RCCL robot simulator interface was replaced by a program called the "RCCL interface" that executed on the QNX based PC. The operation of the RCCL interface is very similar to that of the RCCL robot simulator. First, the RCCL interface waits for an RCCL program to establish a connection (via the simulator interface). As soon as the connection is established, the RCCL interface begins to exchange data packages with the RCCL program. The RCCL interface decodes the data packages and forwards the relevant information to the low level control program (QRobot). In the same manner, the RCCL interface gathers data from the low level control program, encodes them into data packages, and sends them back to the RCCL program running on the UNIX workstation. By using the RCCL interface, a user is able to run a regular RCCL program as though he is using the RCCL simulator; however, the RCCL interface actually controls the robot using QRobot on the QNX PC. The standard RCCL box demo program, and others, were successfully executed.

While use of the RCCL interface proved successful, there are some drawbacks to this method. Two computers are still required (the UNIX workstation which runs RCCL and the PC which runs QRobot). Normally, communication over an Ethernet network would be cause for concern in an application which requires deterministic response. However, this problem was overcome by using an isolated Ethernet network between the UNIX workstation and the QNX PC (a simple 10baseT crossover cable was used to connect secondary Ethernet interfaces between the two computers). Since the high level task space robot program is running on the UNIX workstation, sensor data that is used in trajectory planning must be passed to the UNIX workstation as opposed to simply being retained on the PC.

If the high level robot control library (RCCL in this case) was executing on the same computer as QRobot (*i.e.*, the QNX based PC), all of the disadvantages of the simulator service interface technique would be addressed. Hence, porting RCCL to QNX seemed like a good solution. This idea was not pursued by us since John Lloyd, one of the authors of RCCL, tried unsuccessfully to port RCCL to QNX himself. The primary problem here is that RCCL depends on some UNIX functionality (such as shared address space between threads) which QNX does not support fully.

High Level Software Development for ARCL Connection: ARCL, developed by Peter Corke, is a robot control library which provides some of the functionality of RCCL (though in a much more limited fashion); however, its architecture makes it seem better suited for porting to a PC-based platform. ARCL is composed of three major parts that are listed below. The reader is referred to [23] which describes ARCL in more detail.

API	Application program interface and math support Platform independent Manipulator independent
AKI	Kinematic interface Platform independent Manipulator dependent
AMI	Machine interface Platform dependant Manipulator and controller dependent

Since the AKI for the PUMA 560 is already implemented in ARCL, our first assessment revealed that the only major effort in porting ARCL to QNX would be porting the AMI. The first step in porting ARCL was to obtain the robot and platform independent AMI (called the “offline” UNIX AMI) working under QNX. This AMI doesn't support any real time functionality or low level control, it simply dumps the trajectory generator output to a file. The file can then be analyzed for correctness. Once a QNX specific makefile was written, and some minor adjustments due to differences in the implementation of variable parameter list functions on UNIX and QNX were made, the trajectory generator gave the expected results while running on QNX.

The two major functions of the AMI are to provide the real time capabilities (*i.e.*, calling the trajectory generator at a certain fixed frequency) and to provide communication with the low level robot controller. For communication with the low level control, a client/server communication structure was used. For this application, ARCL's trajectory generator is the client and QRobot (the low level control program) is the server. For robust communication between the AMI and QRobot, we decided to use the client/server classes already developed in our QMotor environment. Unfortunately, the client/server classes were implemented in C++ while ARCL was implemented in C. This meant that either C to C++ interfacing functions had to be written, or ARCL had to be ported to C++. The latter solution seemed to make more sense. While compiling a C program as a C++ program should be trivial, this was not the case with ARCL. C++'s stringent type checking conflicted with some programming techniques used in ARCL, so almost all ARCL modules had to be modified to make them type consistent.

Calling the trajectory generator at a certain frequency is actually an easy task in QNX. The timer and proxy mechanisms in QNX can be used to wake up the trajectory task. However, ARCL used a structure similar to RCCL, in that both the trajectory generator and the user program tasks had to share the same address space. One can remedy this address space problem with one of the following approaches.

1. Link the trajectory generation code into the user program, thereby producing only one program to provide a thread of execution for the trajectory generator. This method can be viewed as co-operative multitasking and is a very poor solution. The user program must call the trajectory generator at the correct frequency regardless of any other tasks that the user program must perform. This type of operation is impossible if the user program blocks for any reason (waiting for user input, disk activity, *etc*.) While this method was not a satisfactory solution, it was the simplest, and did allow us to move the robots from an ARCL program via QRobot.
2. Use threads. Threads are actually multiple instances of the same process that share the same address space (data) but have different stacks. Unfortunately, while QNX does provide a *tfork()* function, which spawns a thread, the support for threads is minimal. Some QNX library functions are thread-safe, some are not, and there is no documentation as to which ones are thread-safe and which are not. If a non thread-safe function is called, the trajectory generator and user program could crash, or worse yet, produce incorrect results with no indication of a failure. Since the trajectory generator only uses math functions, it should be possible to avoid these problems, so this is the next solution that will be pursued.
3. Use tasks with separate address space and use interprocess communication. Shared memory would be the best IPC mechanism for this application; however, there is a lot of preparation involved in getting two processes to share memory. A shared memory segment is not the same as shared address space. When threads share address space, all threads can simply access variables by name. When shared memory is used, one task must create the shared memory, and the rest of the tasks connect to it. All tasks must then agree where in shared memory the shared variables are placed, and the variables must be accessed only through special pointers to the shared memory segment. An alternative would be to write a shared memory manager that would effectively provide a shared address space. This approach should be avoided if possible since this is a complicated task that provides operating system functionality. This course will only be pursued if the thread implementation fails.

With the AMI for QNX functioning (though only in method 1 described above), some simple ARCL programs were implemented. These revealed some limitations of the current implementation of ARCL. First, it was not possible to specify a desired position for the robot in joint space. A look at the ARCL source code revealed that this is partially implemented, but not completed. Some modifications to ARCL were made to allow specification of desired position in joint space, but it was not possible to move to a target in joint space with a desired joint velocity. Rather, the desired time of execution had to be specified. Furthermore, the ARCL trajectory generator simply generates a stream of setpoints and sends them to the controller, without checking to see if the robot is actually moving. While one can make some arguments for this approach, we believe that it would be preferable for the trajectory generator and user program to be given feedback during a move.

7 Conclusion

Our work for the U.S. Department of Energy requires an inexpensive robot control system capable of interfacing to ISA and PCI bus compatible sensors and the ability to modify the control algorithm at the lowest level. Attempts to use RCCL and ARCL have revealed that there are no currently supported general robot control systems that meet our requirements. Previous efforts such as RCCL and ARCL provide partial solutions, but these control environments were developed at a time when the technology was not available to allow a clean implementation of such a system. Both RCCL and ARCL use some object oriented design, yet they do not use object-oriented development tools (*i.e.*, C was used for implementation).

The result of the work presented in this paper is an Intel PC-based robot control system. The low level robot control program (QRobot) works well, as the plots of tracking error have shown. Though the RCCL simulator interface and a preliminary QNX port of ARCL both worked to some extent, and can be used for simple robot tasks, neither is a satisfactory, robust solution. Hence, the next phase in this project is the specification, design, and implementation of an object oriented high level robot control system which would provide functionality similar to ARCL and RCCL. Implementation in C++, an object oriented language, will make the system more portable, maintainable, and extensible. C++ is chosen over other object oriented languages because of performance considerations. C++ programs can be as fast and efficient as C programs, whereas other object oriented programming languages such as Java and Smalltalk are much slower. In addition, the new system will be able to run on QNX as a native program, so that only one PC will be required to control the robot. Based on our previous experience with control implementation, we believe that modern Pentium CPUs are powerful enough so that one PC could actually control multiple robots.

No robot control system is truly complete without a 3-D robot simulator. A simulator allows users to test robot control programs without risking damage to the actual manipulator. Hence, the new robot control system will include an OpenGL [24] based 3-D robot simulator, capable of running on any platform that supports OpenGL (including Silicon Graphics workstations and PCs running Windows 95 and Windows NT). Work on the simulator has already begun, and a screen shot is shown in Figure 10. Currently, the simulator only supports the PUMA 560, but it will be made extensible to any robot which has a D-H representation.

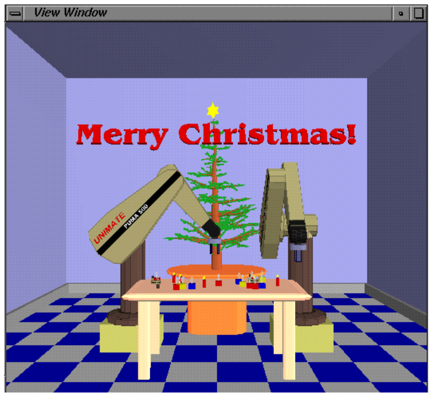


Figure 10 - 3D Robot Simulator

References

[1] http://ece.clemson.edu/iaal/doeweb/doeweb.htm (web site describing this D.O.E. project).

[2] J. Lloyd and V. Hayward, "RCCL/RCI System Overview", McGill Research Centre for Intelligent Machines, McGill University, Montreal, Quebec, Canada.

[3] Unimation Inc., Danbury, Connecticut, "500 Series Equipment and Programming Manual", 1983.

[4] http://www.moonstar.com/FileLib/pdp/faq (web page describing LSI-11/2 processor).

[5] P. Corke, "The Unimation Puma Servo System", CSIRO Division of Manufacturing Technology, Preston, Australia.

[6] http://ece.clemson.edu/crb (Clemson Mechatronics web site).

[7] N. Costescu, "Telerobotic Inspection of Toxic Waste Containers Using a Dexterous Manipulator", Masters Thesis, Clemson University, 1994.

[8] J. Duffie, "Experimental Study of Tracking Control for Robot Manipulators", Masters Thesis, Clemson University, 1993.

[9] D. Reynolds, "Experimental Study of Impedance Control for Robot Manipulators", Masters Thesis, Clemson University, 1993.

[10] J. Lloyd, "Implementation of a Robot Control Development Environment", Masters Thesis, McGill University, Dec.1985

[11] J. Lloyd and V. Hayward, "RCCL/RCI Startup and Installation Guide", McGill Research Centre for Intelligent Machines, McGill University, Montreal, Quebec, Canada.

[12] P. Corke, P. Dunn, and R. Kirkham, " An ARCL Tutorial", CSIRO Division of Manufacturing Technology, Preston, Australia, 1992.

[13] http://www.qnx.com (QNX web site).

[14] D. Hildebrand, " An Architectural Overview of QNX", *Proceedings of the Usenix Workship on Micro-Kernels & Other Kernel Architectures*, Seattle, April, 1992.

[15] http://www.cs.cmu.edu/~deadslug/puma.html (TRC robotics web page).
Trident Robotics and Research, Inc., 2516 Matterhorn Drive, Wexford, PA 15090-7962, (412) 934-8348, email: robodude@cmu.edu

[16] N. Costescu and D. Dawson, "QMotor 2.0 - A PC Based Real-Time Multitasking Graphical Control Environment", Submitted to the *American Control Conference,* Philadelphia, Pennsylvania, June 1998.

[17] Quanser Consulting, 102 George Street, Hamilton, Ontario, CANADA L8P 1E2, Tel: 1 905 527 5208, Fax: 1 905 570 1906, http://www.quanser.com/products/multiq.htm.

[18] Precision MicroDynamics Inc., 3-512 Francis Ave., Victoria, BC, Canada, V8Z-1A1, Tel: 250-382-7249, Fax: 250-382-1830, http://pmdi.com/ind_3ax.html.

[19] N. Moreira, P. Alvito, and P. Lima, "First Steps Towards an Open Control Architecture for a PUMA 560", *draft.*

[20] QSSL, Corporate Headquarters, 175 Terence Matthews Crescent, Kanata, Ontario K2M 1W8 Canada, Tel: +1 800-676-0566 or +1 613-591-0931, Fax: +1 613-591-3579, Email: info@qnx.com.

[21] http://ece.clemson.edu/crb/research/qmotor/qmotor2.htm (web page describing QMotor 2.0).

[22] R. Voyles, Department of Computer Science and Engineering, 4-192 EE/CS Building, 200 Union Street S.E., Minneapolis, MN 55455-0159, Tel: (612) 624-8306, Fax: (612) 625-0572, voyles@cs.umn.edu, http://www-users.cs.umn.edu/~voyles/index.html.

[23] P. Corke, "The ARCL Robot Programming System", CSIRO Division of Manufacturing Technology.

[24] J. Neider, T. Davis, and M. Woo, "OpenGL Programming Guide", Addison-Wesley Publishing Company, New York, 1993.