RWRCCL: A New RCCL Implementation Using Real-Time Linux And A Single CPU

Matthew R. Stein
Roger Williams University
Bristol, Rhode Island 02809
mstein@rwu.edu

Abstract – **John Lloyd and Vince Hayward completed the Robot Control C Library (RCCL) at McGill University in the mid to late 1980's for a variety of robots. This architecture made the best use of computational resources at the time, primarily Unix Workstations, in combination with the robot's original control computer. Because today's desktop computers far exceed the computational power of those available in the 1980's, it is now possible to accomplish all the necessary computations on a single CPU. This paper describes a new control architecture developed at Roger Williams University based on RCCL but using a single Intel CPU computer and a Mark V Automation TRC004 interface card.**

Keywords-PUMA; RCCL; RTLinux; Computer Control

I. *INTRODUCTION*

Unimation® PUMA robots have been popular and omnipresent in academic research labs around the world. The closed system architecture of these robots caused many researchers in the 1980's to deconstruct the PUMA's control system for the purposes of circumventing the limitations imposed by the architecture. One notable system which did this was the Robot Control C Library (RCCL), originally developed by Vincent Hayward at Purdue University[1] and then further developed and extended by John Lloyd at McGill University[2,3]. RCCL provides a C language API for all aspects of robot control above the joint-servo level, including trajectory generation and the coordination of multiple robots. RCCL has not been supported for several years, but the last official release, RCCL 5.0, is still available by anonymous ftp[4].

By the late 90's, inexpensive desktop PCs with remarkable computational power became available. These PCs made it possible, for the first time, to contemplate performing all computational tasks necessary for robot control on a single CPU. Joint-level servo is relatively easy to develop on the PC, and has been implemented at numerous institutions. However, necessary control functions do not stop at servo control. Full functionality requires trajectory generation, task scheduling and homogeneous transform manipulation. In the opinion of this author, the RCCL system performs these latter functions remarkably well. So rather than re-write the higher-level control functions we attempted to update RCCL to a new underlying control architecture.

The new architecture uses a single PC to perform both low and high level control. This requires bypassing the original LSI11-based control computer and directly accessing the analog servo amplifiers and feedback control signals of the robot. Richard Voyles [5] implemented a hardware solution for accessing these signals as part of his dissertation work at Carnegie Mellon University. His spin-off company, Mark V Automation Corp.[6], sells a two-board combination (The TRC004/006) that allows a PC direct access to internal control signals through the EISA bus. The TRC004 also has A/D and D/A converters; so that all control functions can be accomplished with I/O port read and write operations.

The original RCCL system was developed for Unix and C, most prominently on Sun, DEC and IBM workstations. Unix operating systems for the PC were slower to come, but in the last few years Linux has come of age in ease of installation and hardware support. The original RCCL system required a Unix OS patch and re-compile to support real-time functions. Real-Time Linux (available via anonymous ftp [7]) provides the equivalent function.

With all the necessary pieces in place, we endeavored to upgrade the RCCL system to Real-Time Linux on a single PC using the TRC004/006 as the interface between the PC and the PUMA robot. This development was modestly successful and we are offering the new RWRCCL system "Roger Williams Robot C Control Language" free via anonymous ftp [8]. This paper will describe the development and present technical details of the new implementation. Section 2 will present the new control architecture with reference to the original RCCL system, Section 3 the new Joint-Level Control. Section 4 discusses the performance of the system and Section 5 additional components of the RWRCCL system. In Section 6 we will present summary and conclusions.

II. THE NEW CONTROL ARCHITECTURE

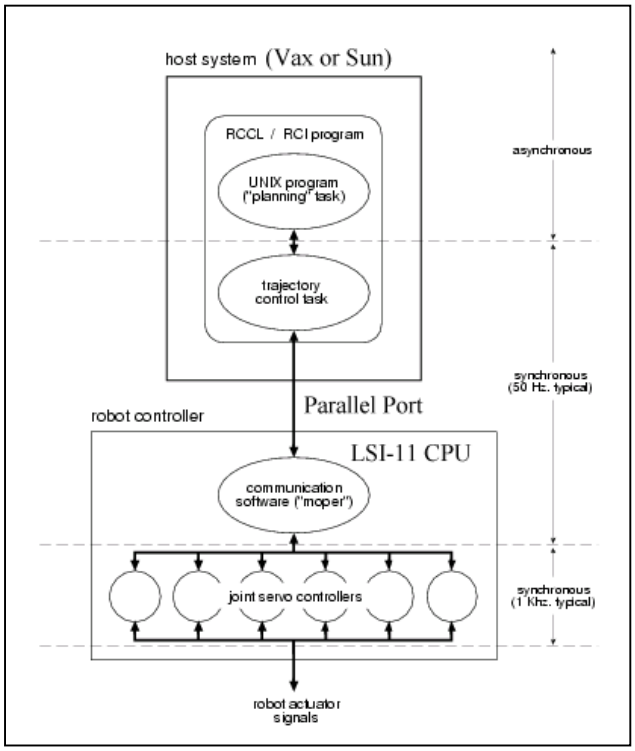

Figure 1 Original RCCL architecture from included documentation (with added annotations).

Figures 1 and 2 show the control architecture for the original RCCL and new RWRCCL systems respectively‡. As the figures demonstrate, the original system used the LSI-11 control computer and the manufacturer supplied joint servo control cards. Both are omitted from the new architecture, replaced by "direct" access through digital and analog interface circuitry of the TRC004. A consequence of this architecture is the host system assumes responsibility for joint servo level control in addition to the planning and trajectory control tasks.

The synchronous processing in the original system accomplished two purposes as Figure 1 suggests. The first is synchronous communication with the host via parallel port channel. Around every 20 milliseconds the trajectory control task on the host and the communication software on the LSI-11 would synchronize and exchange information. This function is replaced by the direct communication with the robot hardware. The second function is trajectory control, providing smooth and coordinated motion of the joints and requiring a sense of real-time. This function is not replaced by the joint level servo task.

A schematic of the RWRCCL software architecture is shown in Figure 3. To emulate the real-time interrupt for the trajectory generator, we use an asynchronous signal from the joint-level servo process. The trajectory generation process will be suspended awaiting this signal, and will execute approximately every 30 milliseconds.

‡ Original figures are from the Hardware Installation Notes section of the RCCL documentation.

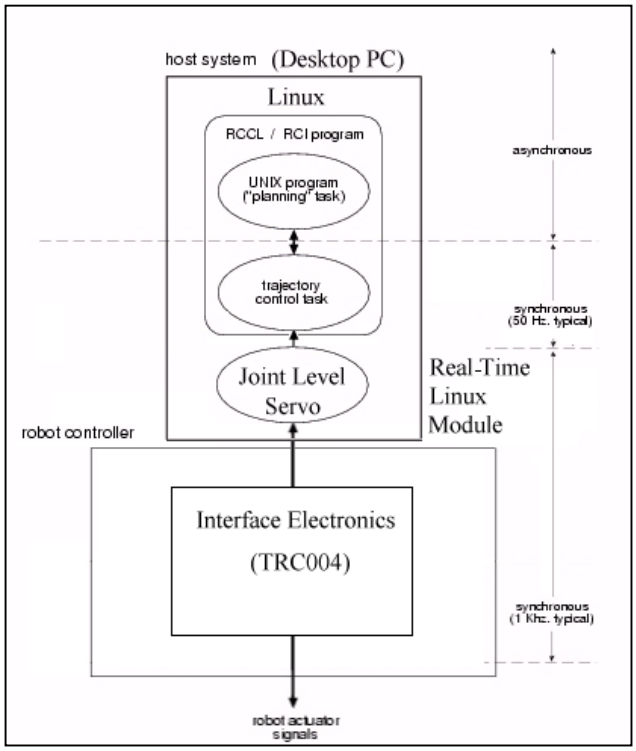

Figure 2 New RWRCCL architecture, modified from RCCL documentation.

Note that the shared memory blocks between the trajectory and planning levels (the HOW, RBT, etc. structures) are the same as the original RCCL system. On each trajectory control cycle, two functions (puma_input and puma_output) perform the reading and writing functions to the Joint-Level Servo (JLS) process via shared memory.

The asynchronous level shown in Figures 1 and 2 are regular Unix processes. These are user programs written in 'C' or 'C++' that call functions from a library of routines that support robot control. The Linux kernel schedules execution of these programs in a time-sharing environment. Significantly, the top portion of Figures 1 and 2 are similar; indicating no detectable difference between the architectures at the user level.

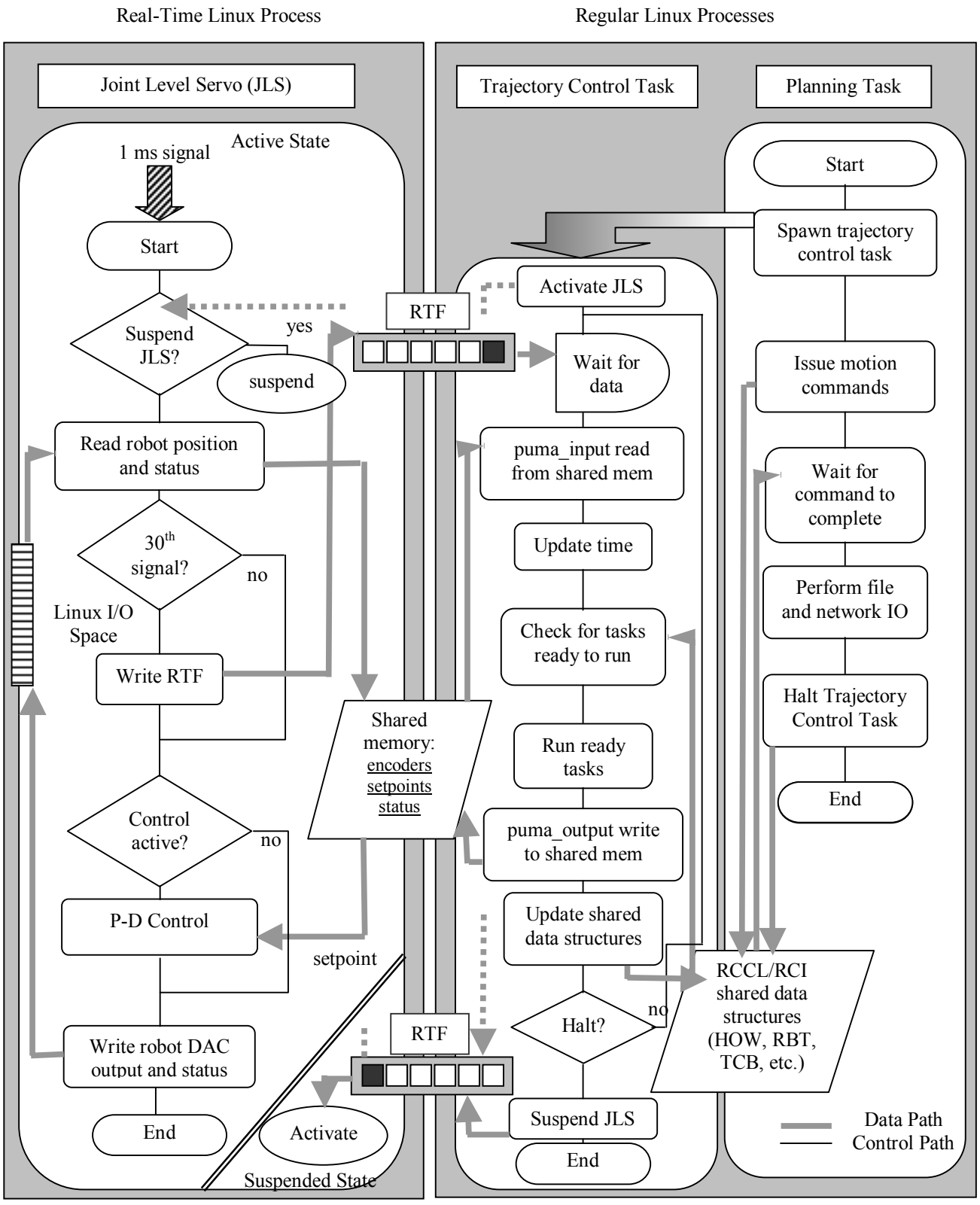

Figure 3 Diagram of software architecture of the RWRCCL system

III. The Joint-Level Control System

Real-Time Linux provides the facilities for creating a periodic task and controlling its execution. To ensure the real-time thread never blocks, all communication with asynchronous (regular Linux) processes occurs through Real-Time FIFO (RTF) queues. The real time process can write to these queues without blocking and can perform read operations if there are data available. Part of the real-time process can be initiated when data are available on an RTF, thus making it possible to initiate, control the execution and suspend the real-time process from a regular Linux process through appropriate writes to the RTF.

The real-time process is a module that must be loaded into the kernel before execution of an RWRCCL program. The process will remain suspended until activated, so there is no significant penalty in loading the modules at boot time. The periodic process executes at 1 kHz to mimic the frequency of the original joint servo control cards. Our hardware is a 600MHz Pentium III on 133 MHz bus with 128 MB RAM.

We developed the closed loop control making heavy reference to the technical report "The PUMA Servo System"[9] by Peter Corke [10]. Initially, we verified that the dynamic model parameters presented in the report reflected the dynamics of our PUMA, and these did remarkably well. Using MatLAB, we built a state-space controller based on the verified model, but were disappointed to find the floating-point operations caused clock overruns of the Joint-Level Servo process. We compromised with a straightforward P-D control with some filtering, as shown in Figure 4. More recent versions of RealTime Linux offer support for floating point operations, and a 600MHz PIII can no longer be considered a "fast processor", so there is a possibility for more sophisticated control with improved hardware.

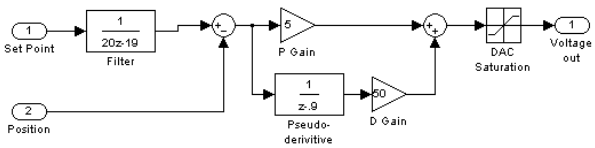

Figure 4 P-D controller for joint level servo

The performance of the P-D control is not particularly good, as can be seen by the graph of Joint 2 control as shown in Figure 5. We see the P-D control lags the Setpoint and filtered Setpoint curves and has a steady-state error. In the motion shown the steady-state error is 33 encoder counts or approximately 0.15 degrees. Steady-state error in the P-D control is determined by friction and gravitational loading and therefore will be position dependent. For a large motion (e.g. 30 degrees) the trajectory generator changes the Setpoint stepwise as can be seen in Figure 5. This introduced some notable jitter in the robot motion, and we reduced this by adding the setpoint filter shown in Figure 4.

We are relatively satisfied with this performance, considering our imprecise applications of online painting and education. Nothing we do with our robot requires great accuracy, so we are able to live with some controller "slop" and some "slumping" due to gravitational biasing. We acknowledge that this control may not be adequate for other users with applications requiring greater accuracy.

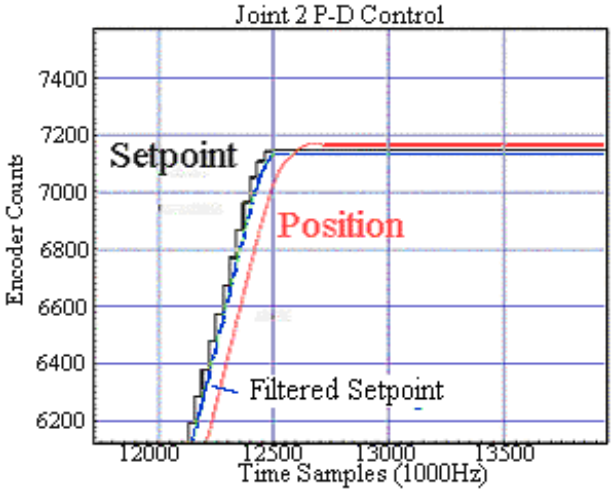

Figure 5 Performance of Joint 2 P-D controller for 30-degree motion.

IV. System Performance

User programs written for the original RCCL system will execute on the RWRCCL system without modification. For example, all of the demo programs for the PUMA560 included with the RCCL distribution were executed using RWRCCL after a simple re-compile.

As shown in Figure 2, the trajectory generation process is a regular Linux process subjected to pre-emptive scheduling. The process blocks on a read to a real-time FIFO and returns from this read when there are data to be read, as shown in Figure 3. As a consequence, the trajectory generation process is not truly synchronous and this has some small but noticeable effects on the robot motion.

We find occasional variation in the completion time of commands as the robot may pause slightly during a motion. We also find we can worsen this effect by loading the processor with time-consuming disk writes or screen refreshes. Again, utilizing a faster machine may alleviate some of these problems.

V. Additional Items

A. Robotsim

A useful robot simulator **robotsim** is included in the original RCCL distribution. This provides a faithful simulation of an RCCL program controlling a robot via RCI. We modified **robotsim** to execute under the RWRCCL configuration. We again use a RealTime Linux module to generate real-time FIFO messages to the **robotsim** process. This is a kernel module similar to the Joint-Level Servo module but

performing no servo control. Although both modules can be resident in the kernel, RealTime Linux by default allows only one real-time process at a time. There are third-party preemptive scheduling modules available, but we did not experiment with these. It is therefore not possible to run the simulator and the robot controller simultaneously. After some porting of the simple graphics to current X-windows libraries, all of the upper-level functions of **robotsim** are unchanged.

B. Teachdemo

An interactive program **teachdemo** simulates the teach pendant and allows live, command-line control of the robot. The **teachdemo** program compiled and functioned in the RWRCCL system without modification. However, the actual teach pendant was connected via serial line to the LSI-11 processor, so this connection required modification. We re-wired the teach pendant through the control chassis to the 9-pin serial port of the Linux machine (/dev/ttys0). The teach pendant sends straightforward two-byte packets on a 9600 baud serial line when buttons are pressed. Similarly, two-byte packets can be sent to the teach pendant to set the state of LEDs or cause a message to appear on the single-line display. Currently, we can exchange command packets with the teach pendant asynchronously. To date we have not integrated serial port communication into the **teachdemo** program, but expect this will be straightforward.

C. Calibration

For reasons not entirely clear to us we are unable to use the TRC004 A/D converters to read the joint potentiometers. It may be we have damaged our TRC004 card; because analog signals presented to the A/D appear correct when measured by hand. We have not implemented the dynamic calibration function of the PUMA because of this limitation. This function normally allows the PUMA to calibrate itself from any position in its workspace (the sole function of the potentiometers). We instead rely on manual calibration, where we move the robot to a known position by hand and load the encoders with corresponding values using the **primecal** program.

D. Root users

The TRC006 card resides in reserved I/O space of the Linux operating system. Since programs must read and write to these ports, currently all RWRCCL program must be run as root (superuser). There are probably a workarounds for this, but we have not explored these. We have no qualms with allowing a few users to log in as root to run programs, but other institutions may not be as casual.

VI. CONCLUSION

PUMA robots can still frequently be found in academic labs. A shoestring budget is one reason we (and perhaps others) are still working with PUMA robots, these often being donated and/or used equipment. We imagine researchers like ourselves are frequently searching for the least expensive way to make a robot operational.

The solution described here is suitably cost effective, especially if one already has an old PUMA robot. It requires purchasing only the Mark V Automation TRC004/006 cards (about $1200 at the time of writing) and one decent performance PC (<$1000). Not to suggest these are entirely trivial hardware expenses. As a point of comparison, at the time of this writing an equivalent Galil™ motion control card for six axes lists at about $2000 (including cabling), and this only provides joint-level servo.

All of the software required is free via anonymous ftp. Linux is widely available and a suitable version of RTLinux (a kernel modification) is still offered for free. We are using Slackware 8.1 Linux [11] with the 2.4.4 Kernel and RTLinux3.2. Although not for the novice, installation of RTLinux is no more complicated that re-compiling the Linux kernel (note RealTime Linux version must match the kernel version). The RWRCCL is also free from the author, and we have endeavored to make installation no more complex than installation of the original RCCL system.

As Intel-based personal computers have become both more powerful and less expensive, it seems the trend towards using them for more ambition real-time control will continue. Many researchers have used PCs for joint control, but the higher-level software is also essential. We believe it is hard to find a high-level programming system better than RCCL, so we have incorporated this functionality on top of an inexpensive, PC-based servo architecture. Although our solution has imperfections, we are presenting it here in the hopes that it proves useful to other researchers experiencing similar needs.

ACKNOWLEDGMENT

The author would like to thank the Roger Williams University Research Foundation for funds supporting this project and for a Course Release Grant in the Fall 2002 semester for preparation of this manuscript. Grateful thanks are also due to undergraduate students Christopher P. Madden and Stephanie Lemmo for their contributions to this project. The robot described in this paper is a donation from NASA JPL and the author would like to thank Paul S. Schenker, Eric Paljug and many others for making this possible.

REFERENCES

[1] Vincent Hayward and Richard P. Paul, "Robot manipulator control under unix: RCCL A Robot Control `C' Library". International Journal of Robotics Research, MIT Press, Vol. 5(4), 1987, pp. 94-111.

[2] John E. Lloyd, Mike Parker, and Rick McClain, ``Extending the RCCL programming environment to multiple robots and processors". IEEE International Conference on Robotics and Automation Philadelphia, Pa., April 24-29, 1988, pp. 465-469.

[3] John E. Lloyd and Vincent Hayward, "Multi-RCCL User's Guide", McGill University, 1992, ftp://ftp.cs.ubc.ca/pub/local/RCCL/rcclGuide.ps.Z

[4] http://www.cs.ubc.ca/spider/lloyd/rccl.html

[5] http://www-users.cs.umn.edu/~voyles/index.html

[6] http://pages.prodigy.net/tridentrobotics/index.html

[7] http://www2.fsmlabs.com/3.2-free.html

[8] ftp://pumapaint.rwu.edu/pub/rwrccl5.1.tgz

[9] http://www.cat.csiro.au/cmst/staff/pic/docs/pumaservo.ps

[10] http://www.cat.csiro.au/cmst/staff/pic/pic.html

[11] http://www.slackware.com