**Proceedings of DETC'02**
**ASME 2002 Design Engineering Technical Conferences and**
**The Computers and Information in Engineering Conference**
**Montreal, Canada, September 29-October 2, 2002**

# DETC2002/EUC-34506

## EMBEDDED REAL-TIME LINUX FOR CABLE ROBOT CONTROL

Frederick M. Proctor and William P. Shackleford
Control Systems Group
National Institute of Standards and Technology
100 Bureau Drive, Stop 8230
Gaithersburg, MD 20899-8230 USA[1]

### ABSTRACT

Linux is a version of the Unix operating system distributed according to the open source model. Programmers are free to adapt the source code for their purposes, but are required to make their modifications or enhancements available as open source software as well. This model has fostered the widespread adoption of Linux for typical Unix server and workstation roles, and also in more arcane applications such as embedded or real-time computing.

Embedded applications typically run in small physical and computing footprints, usually without fragile peripherals like hard disk drives. Special configurations are required to support these limited environments. Real-time applications require guarantees that tasks will execute within their deadlines, something not possible in general with the normal Linux scheduler. Real-time extensions to Linux enable deterministic scheduling, at task periods at tens of microseconds.

This paper describes embedded and real-time Linux, and an application for distributed control of a Stewart Platform cable robot. Special Linux configuration requirements are detailed, and the architecture for teleoperated control of the cable robot is presented, with emphasis on the resolved-rate control of the suspended platform.

### INTRODUCTION

Linux is a version of the Unix operating system written by Linus Torvalds in 1991 while a student at the University of Helsinki in Finland. version 1.0 was released in 1994 and has grown with assistance from programmers across the Internet [1]. It includes features commonly expected from modern operating systems, including multiprocessing, multitasking, virtual memory, shared libraries, demand loading, memory management, and TCP/IP networking.

Linux is distributed according to the open source model [2]. Programmers are free to adapt the source code for their purposes, but are required to make their modifications or enhancements available as open source software as well. This model has fostered the widespread adoption of Linux for typical Unix server and workstation roles, and also in more arcane applications such as embedded or real-time computing.

Embedded applications typically run in small physical and computing footprints, often without keyboards, mice or monitors, and usually without rotating media such as hard drives or CD-ROMs that could not withstand harsh environments. Programmers can configure Linux to run without any of these devices, and there are several popular embedded Linux distributions that come pre-configured and include useful tools for customization. Common customizations include adding flash memory storage to replace rotating media hard disks, and replacing the memory- and disk-consuming X Windows graphics component with stripped-down versions.

Real-time applications require guarantees that tasks will execute within their deadlines, typically with sub-millisecond periods. The normal Linux scheduler is optimized for fastest average response, but does not guarantee task execution by deadlines. Several groups have made real-time modifications to Linux. The New Mexico Institute of Mining and Technology developed Real-Time Linux (NMT RTL) [3]. The Department

[1] No approval or endorsement of any commercial product by the National Institute of Standards and Technology is intended or implied. Certain commercial equipment, instruments, or materials are identified in this report in order to facilitate understanding. Such identification does not imply recommendation or endorsement by the National Institute of Standards and Technology, nor does it imply that the materials or equipment identified are necessarily the best available for the purpose.

This publication was prepared by United States Government employees as part of their official duties and is, therefore, a work of the U.S. Government and not subject to copyright.


Copyright © 2002 by ASME

of Aerospace Engineering of the Polytechnic Institute of Milan developed the Real Time Application Interface (RTAI) [4]. These versions of real-time Linux are available free as patches to the Linux source code, or through commercial vendors who provide documentation and support.

Our application, control of a cable robot, had both embedded- and real-time requirements. We chose Linux as the basic operating system due to our familiarity with it as a desktop scientific computing and file server platform, its low cost and high performance. Many hardware platforms were considered, which we limited to those based on the Intel Pentium microprocessor for consistency with the Linux desktop computers we already used for software development. We were further limited to those platforms designed for harsh environments, which for the cable robot application included large-scale blast-media paint removal. Choices range from single-board computers in passive backplanes, to more integrated form factors such as CompactPCI [5] and PC/104 [6]. We chose PC/104 primarily due to our packaging constraints.

The first part of this paper discusses the use of Linux for embedded real-time applications, while the second part describes the robotics application itself. Readers interested in Linux development need not read the application section in depth, while those readers with robotics interest need not thoroughly understand the Linux configuration issues in the early sections.

## CONFIGURING EMBEDDED LINUX

Considerable flexibility is provided with Linux to eliminate unneeded parts of the kernel (the operating system proper) and system- and user programs in order to reduce memory- and disk consumption [7]. Indeed, many commercial products run Linux internally and do not resemble desktop computers at all, for example, set-top boxes, television video recorders, personal digital assistants (PDAs), and game consoles.

### Embedded Linux Distributions

It is possible to use a full-blown multi-user Linux system as an embedded system, but it typically provides unwanted services (e.g., password login) and takes more memory and mass storage than may be available. Many Linux distributions streamline the system considerably, both reducing the time to boot and resource consumption; a web list is maintained at [8]. We selected BusyBox Linux [9], distributed free under the open source model.

### Kernel Customization and Development Cycle

Linux distributions (e.g., Red Hat, Mandrake and SuSE) come with pre-configured kernels set up for most commonly-encountered configurations. These distributions also provide the source code to the kernel (in C and assembly), so that users can customize what is included and excluded. Most Linux users need not change any kernel configuration, although some choose to leave out features they know they won't need, such as Plug-and-Play, networking, or sound support. A few modify the source code themselves as hobbyists, researchers or out of curiosity. In our embedded development, we did not need to modify the source code, although we did need to change the configuration. The steps involved are relatively straightforward:

1. run the graphical configuration tool and select what is to be included or excluded. Linux provides the `make` utility to automate this, and the kernel source code comes with a makefile with all the rules necessary for configuration.
2. Recompile the kernel, again using `make`.
3. Install the kernel on the target system.

Details have been omitted to keep the discussion focused on the process. The documentation that comes with embedded Linux distributions describes the installation and configuration processes in adequate detail.

Steps (1) and (2) can be done on a system other than the target embedded system for speed and convenience. The result of these steps is the executable kernel file, typically on the order of a megabyte or so in size, that can be put on a floppy and taken to the target embedded system for step (3). The target system can be fitted with peripherals such as floppy disk drives, hard disk drives or CD-ROMs during the development stage to make this process easier. Once the kernel and application code are stable and installed, these peripherals can be removed.

### Diskless Operation

Perhaps the most common embedded customization is the replacement of rotating media hard disks with solid-state read-write storage [10]. These solid-state devices range in size from a few megabytes up to hundreds of megabytes and are suitable for storing the full Linux operating system and any user application code. One example is Compact Flash [11], solid-state media with a built-in IDE interface that serves as a direct replacement for normal IDE hard disks. Another example is DiskOnChip [12], which contains PC BIOS replacement code for dealing with the chip as a normal hard disk. However, these routines are useless with Linux, which does not use the PC BIOS, so special drivers for DiskOnChip devices are required. Recent releases of the Linux kernel (2.4 or higher) include these drivers as part of the Memory Technology Devices (MTD) subsystem. We used DiskOnChip in our system as it was provided with the PC/104 microprocessor module we selected.

Flash technology is limited in the amount of writes that can be done. Several solutions to this problem exist. Linux can be configured to use various file systems that sit atop the raw media. The typical Linux file system is ext2, and DOS FAT can be configured for compatibility with older PC-based drives. File systems specially designed to do "wear leveling" are available. Wear leveling spreads out writes across the underlying physical media transparently, so that repeated writes of a single file will actually take place at different spots in flash. One example is the Journalling Flash File System (JFFS) [13]. The DiskOnChip device builds wear leveling into the on-chip driver (TrueFFS), and so can accomplish wear leveling

 Copyright © 2002 by ASME

with any file system. With these techniques, flash media can last far longer than the expected lifetimes of most embedded applications, on the order of hundreds of years depending on the amount of data written.

Linux requires that some directories be writeable during normal operation, such as those holding log files. In embedded operation these may not need to persist between reboots. In this case, these directories can be mounted onto RAM disks, directing writes away from flash.

### Booting

PCs normally boot by running the boot loader located on the first sector of the hard disk. Various boot loaders exist that can boot Linux, most commonly SYSLINUX [14], LILO [15], or GRUB [16]. These can be configured to boot one of several operating systems loaded onto partitions on one or more hard disks, enabling boot-time selection choices for Linux or other operating systems.

LILO (the Linux Loader) is one of the most popular boot loaders for Linux. LILO was chosen for the cable robot project as it was both already part of the Linux distribution we use (Red Hat Linux) and has support for booting from DiskOnChip devices using the doc-lilo extension to LILO [17].

An initial RAM disk is required when booting from DiskOnChip since the basic BIOS cannot extract data from these devices as it could from direct disk replacements like Compact Flash. The initial RAM disk is a compressed image of the Linux kernel and files necessary for booting that is created by the boot loader (LILO in our case), and later decompressed and loaded into a section of RAM [18]. The RAM disk appears for all intents and purposes like a normal disk, and full bootup continues from this point.

For highly embedded systems with immediate-on requirements, boot time can be reduced drastically by eliminating the BIOS. LinuxBIOS [19] is an open-source project aimed at replacing the normal BIOS with a small amount of hardware initialization and a compressed Linux kernel that can be booted from a cold start in a few seconds. BIOS replacement with LinuxBIOS can be done by burning the LinuxBIOS appropriate for the chipset into ROM and replacing the original BIOS chip; by downloading new BIOS firmware if the BIOS chip allows this; or by using DiskOnChip. We did not have immediate-on requirements and did not port LinuxBIOS to our processor board, since this requires a significant time investment.

It is also possible to configure Linux to boot across the network. This is commonly done on networked computer clusters, where perhaps hundreds of computers run with no local hard disk, keyboard, mouse or monitor. The advantage is that no boot media is required at all (neither disks nor flash), and a single kernel can be used for all cluster processors which makes kernel modification and deployment easier. In non-networked embedded systems this is of course not possible.

## GRAPHICS SUPPORT

Graphics configuration for embedded system is one area that merits special attention. Typically Linux distributions provide the XFree86 implementation of X Windows [20], with bitmapped color graphics, mouse and keyboard input, and window managers. While indispensable for desktop systems, XFree86 consumes tens of megabytes of disk space and memory and may be unsuitable for a small-footprint embedded system. Other alternatives to XFree86 exist that provide all the common functions needed by graphical embedded systems, such as keyboard and mouse input, bitmap graphics, and multiple windows, leaving out more sophisticated functions. Examples include GGI [21], DinX [22], MicroWindows/NanoX [23] and Qt/Embedded [24]. The X Windows server normally available that maps graphics calls to the underlying chipset is not used by these systems. Instead, they map to the hardware using either low-level SVGA graphics functions via svgalib [25], or through the newer Linux Frame Buffer device [26]. The Frame Buffer is a software layer ported to common video chipsets that enables program portability for graphics software that does not need full X Windows functionality. The advantage of the Frame Buffer interface over svgalib is that it supports more colors and higher resolutions than allowed by the SVGA specification. The disadvantage is that it has been ported to fewer boards, and requires additional configuration.

For our controller, we chose Qt/Embedded, a C++ application programming interface that provides common widgets (e.g., labels and pushbuttons) and can be compiled for both X Windows and Frame Buffer implementations (as well as Microsoft Windows and the Macintosh). Having dual X Windows and Frame Buffer compile options allowed us to debug the graphical user interface (GUI) conveniently on our desktop machines running X Windows, and then do the final port to the embedded system quickly.

## CONFIGURING REAL-TIME LINUX

Both versions of real-time Linux described earlier, RTL and RTAI, are patches to the Linux kernel source code that add a real-time scheduler and insert a layer between the original kernel and interrupt sources. This layer defers the handling of interrupts by Linux until the real-time scheduler determines that no real-time tasks are ready to run. Real-time and non-real-time processes are strictly separated from a scheduling viewpoint, so that any real-time process will interrupt any kernel code when ready to run, and the Linux kernel will not preempt any real-time process. Interprocess communication mechanisms such as semaphores, shared memory and FIFOs are used to link real-time and non-real-time code.

The same steps are taken to install either real-time Linux version:

1. patch the kernel source code using the `patch` utility, or obtain a pre-patched kernel. If a pre-patched kernel is used, it will likely not have any embedded customizations done


Copyright © 2002 by ASME

earlier. In this case the embedded customizations should be done to the pre-patched kernel.
2. Recompile the kernel, identical to step (2) for embedded system configuration earlier.
3. Install the kernel, identical to step (3) for embedded systems.

Details have been omitted to keep the discussion focused on the process. The documentation that comes with each RT Linux system describes the installation and configuration processes in adequate detail.

## REAL-TIME DEVELOPMENT

Because Linux processes and real-time processes are scheduled separately, with real-time processes having higher priority, any Linux code (even the kernel) can be preempted by real-time code. For this reason, real-time code cannot make Linux system calls, device driver calls, or otherwise interrupt the flow of Linux processes. For example, real-time code cannot call the `malloc` function to allocate heap storage, since the kernel call that implements this may be in the middle of updating the memory allocation table when interrupted by the real-time process, corrupting the table. It is possible to re-code the Linux kernel so that its system calls can be interrupted by themselves (termed "reentrancy"), but this was not a design goal of Linux' developers as it was with commercial real-time operating system vendors. As a result, real-time Linux requires its own copies of device drivers for hardware resources, and these resources cannot be shared with normal Linux processes. RT Linux drivers for serial communication and various industrial communication buses exist, so this limitation affects the design of the system, and does not impose a burden on the programmer. However, there are far fewer device drivers for RT Linux, so any hardware considered should be checked for available RT Linux drivers.

## THE CABLE ROBOT AND CONTROLLER

In our application, we control a large-scale moving platform suspended by six wire rope cables arranged in a Stewart Platform configuration [27, 28, 29]. The intent of the platform is to carry construction and maintenance equipment, such as paint sprayers or removers, throughout a large work volume typical of ships or aircraft. This is shown in Figure 1. Servo-controlled hoist motors mounted on the platform or support structure lengthen or shorten the cables. Individually, a cable will move the platform in a non-intuitive way. When coordinated properly, all six cables can effect straight-line motion of the platform, indeed any motion desired. Several applications of this configuration, termed the RoboCrane [30], have been built.

The anchor points of the cables to the wall or ceiling must be measured with respect to a ground-based coordinate system. The anchor points of the cables to their hoists on the platform must likewise be measured with respect to a platform-based coordinate system. The controlled position is then the position of the platform origin relative to the ground origin, which moves around as the cables lengthen and shorten. These calibration measurements need only be done once, when the platform is installed at the facility.

Amplifiers with built-in velocity servos power the hoist motors. The amplifiers connect to digital interfaces that provide a serial link over which velocity commands and position feedback are sent. Depending on the configuration, a single serial link may connect to a single digital interface and amplifier serving a single motor, or several serial links may each connect to a digital interface, each controlling several amplifiers and their associated motors. This complicates the communication protocol and reduces the available bandwidth for control, and requires the use of RS-485 multidrop serial signaling instead of the usual RS-232 single-drop serial signaling supported for example by a PC's COM1 port. These signaling types are jumper-selectable on the PC/104 modules.

Figure 1. Cable robots capable of six-degree-of-freedom motion, in an aircraft maintenance application. Hoists on the moving platforms lengthen or shorten cables in a coordinated way to move the platform in an intuitive coordinate system. Computer control calculates the transformations from user coordinates to cable lengths.

### Control Method

The controller implements resolved-rate teleoperation, in which a joystick generates the desired velocity of the moving platform in Cartesian space (X, Y, Z, roll, pitch, and yaw). This desired velocity is transformed into cable speeds through the inverse Jacobian function:

$$W = J^{-1} V \qquad (1)$$

where $W$ is the 6x1 cable speed vector, $V$ is the 6x1 Cartesian velocity vector, and $J^{-1}$ is the 6x6 inverse Jacobian transform matrix that depends on the current Cartesian position of the moving platform. This is an instantaneous relationship. In a sampled system, where some time elapses between successive


Copyright © 2002 by ASME

recalculations of the inverse Jacobian matrix, the cable speeds will be constant during this interval. As a result, the moving platform will accumulate position errors and require correction. Normally the operator would compensate for these errors, which are small, but in our case the Cartesian roll and pitch velocities are forced to zero to keep the platform level. However, as errors accumulate, the platform will go out of level and require some compensation.

It is possible to correct these automatically, since the actual Cartesian position (including roll and pitch) are continually computed by reading the cable lengths from the motor encoders and running these through the forward kinematics function:

$$C = T\,\theta \qquad (2)$$

where $C$ is the actual 6x1 Cartesian position vector, $\theta$ is the 6x1 cable length vector, and $T$ is the 6x6 matrix for the forward kinematic transform. Since actual roll and pitch are known, velocities in the compensating direction can be automatically computed and used in place of the nominal zero values to drive the platform toward level continually.

This method will not work to level the platform in the presence of perturbations that cannot be sensed by the motor encoders, such as cable stretch or slack due to unbalanced platform loading. To compensate for these errors, a level sensor is used to drive the compensating roll and pitch velocities instead of the kinematic leveling just described.

In the case of the Stewart Platform, the inverse Jacobian transformation $J^{-1}$ is closed form. However, the forward kinematic transform $T$ is not closed form, and iterative calculations must be performed to get an estimate of the true Cartesian position $C$. The iterative algorithm requires an initial estimate of the Cartesian quantity it is trying to compute in order to converge. During normal operation, this estimate is simply the last Cartesian position computed, which changes little from cycle to cycle. However, at the beginning, we need a matched pair of cable- and Cartesian positions in order to begin the iterations. This is accomplished by a homing procedure as follows:

1. Select a Cartesian position to be the natural home position $C_H$. This is often the point where the platform rests when idle or where it picks up loads. It may be the ground origin, but need not be.
2. Compute the corresponding cable lengths at the home position $\theta_H$ using the closed-form inverse kinematic transform,

   $$\theta_H = T^{-1}\,C_H \qquad (3)$$

   where $T^{-1}$ is the inverse of the 6x6 forward kinematic transform $T$ from Equation 2.
   Steps (1) and (2) are done once, when the system is set up and calibrated, and saved in a configuration file or otherwise preserved as static system parameters.
3. When starting up the controller from an unknown position, jog the cables individually until they are at their home position $\theta_H$. This can be a tedious process. If possible, the cables can be marked so that their home lengths are obvious.
4. When all cables are at their home positions, signify to the controller that homing has been done. At that time the iterative forward kinematic calculations will have a valid estimate and can maintain an estimate throughout operation.

It is possible to run the platform in a degraded but still useful mode, in which the home position was only approximated. In this case, operator commands to move the platform along a straight line will result in somewhat skewed behavior. However, Cartesian teleoperation in this skewed mode may be more intuitive than individual cable motion, and can be used to bring the platform to its true home position more quickly than by moving individual cables. Once the platform is brought to its true home, step (4) is performed.

If the controller can preserve its last Cartesian position upon shutdown and restore it when starting up later, then the homing procedure need be done only once when the system is first set up and calibrated.

### Controller Description

Our controller is a PC/104-based system with a Geode Pentium-compatible processor, running BusyBox Linux, kernel 2.4.1, with the RTL 3.0 real-time patch. Mass storage is a 96-megabyte DiskOnChip. The graphics system consists of a serial touch screen on an 800x600-pixel display, running the Qt/Embedded window interface. Additional PC/104 I/O modules provide up to 8 serial connections, and some digital- and analog I/O. Portions of the system are shown in Figure 2.

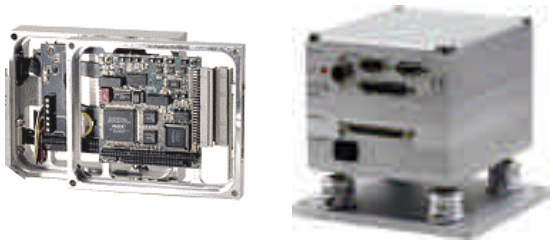

Figure 2. PC/104 modules for computing and I/O, and an assembled stack.

Serial connections to the motor amplifiers vary depending on the applications we have built. In one case we use six dedicated 56-kbps links, one for each digital interface/amplifier/motor triplet. In another, we use two 9.6 kbps links, one for each of two groups of digital interfaces that can each drive three amplifier/motor pairs. Velocity command messages and associated overhead occupy about 30 bytes. In the first configuration, all motor commands can be sent each controller cycle, for a lower bound on the control cycle of about 5


Copyright © 2002 by ASME

milliseconds. In the second configuration, motor commands are sent out in pairs, for a lower bound on the control cycle of about 100 milliseconds. Note that the longer the period, the longer the time interval over which the relationships between Cartesian- and cable velocities will deviate from their exact values at the start of the interval. This will induce more drift in the roll and pitch directions (as well as the others), and cause the platform to go out of level to a greater degree. So, although the control period does not affect stability (velocity servo control is done in the amplifiers), it does affect kinematic fidelity.

The Cartesian position is saved at shutdown and restored upon powerup, so that homing need be done only once at installation time or periodically after maintenance. No other information is written to the flash disk. Using the lifetime model provided by the flash disk's manufacturer, writing this small amount of data daily onto approximately 60 megabytes of free wear-leveled storage yields a lifetime far in excess of the device's quoted 148 years of mean time between failure.

## SUMMARY

Linux is an operating system distributed freely following the open source model, and has been ported to a wide variety of processor architectures. Real-time extensions exist, making it attractive for embedded control. Considerable flexibility in configuration is provided by Linux and embedded/real-time distributions. Flash media can be used in place of rotating media hard disks, and streamlined graphics environments provide sophisticated GUI support with modest resource requirements. A PC/104 embedded platform running embedded real-time Linux was used to control a Stewart Platform cable robot, with serial links for velocity control of distributed motors. Kinematic considerations make homing an issue, but state information is saved between shutdown and startup that reduces the burden on users.

## REFERENCES


1. Linux Online, "What Is Linux?" Web reference: www.linux.org
2. The Free Software Foundation, GNU General Public License. Web reference: www.gnu.org/copyleft/gpl.html
3. Real-Time Linux. Web reference: www.rtlinux.org
4. Real-Time Application Interface. Web reference: www.aero.polimi.it/projects/rtai
5. PCI Industrial Computer Manufacturing Group (PICMG), "CompactPCI Standard," Web reference: www.picmg.com/test/compci.htm
6. PC/104 Consortium, "PC/104 Embedded-PC Modules," Web reference: www.pc104.org
7. Christophe Boyanique, "Reducing a Gnu/Linux Distribution for Embedded Use," Proceedings of the Third Real-Time Linux Workshop, Milan Italy, 27-28 November 2001.
8. LinuxLinks.com, "Embedded Distributions," Web reference: www.linuxlinks.com/Embedded/Distributions
9. Erik Andersen, "BusyBox Linux," Web reference: www.busybox.net
10. Robert Kaiser, "Diskless Embedded Linux Systems," Proceedings of the Third Real-Time Linux Workshop, Milan Italy, 27-28 November 2001.
11. CompactFlash Association, "CompactFlash," www.compactflash.org
12. M-Systems, "DiskOnChip," http://www.m-sys.com
13. Axis Communications, "Journaling Flash File System," developer.axis.com/software/jffs
14. H. Peter Anvin, SYSLINUX. Web reference: syslinux.zytor.com
15. Miroslav Skoric, "The Linux Loader (LILO)," Web reference: www.tldp.org/HOWTO/mini/LILO.html
16. Free Software Foundation, "GNU GRUB (Grand Unified Bootloader)," Web reference: www.gnu.org/software/grub
17. M-Systems, "Using the DiskOnChip with Linux OS," Installation Manual IM-DOC-021, November 2000.
18. Linux Online, "The Linux Bootdisk HOWTO," www.linux.org/docs/ldp/howto/Bootdisk-HOWTO/index.html
19. Los Alamos National Laboratory, "The LinuxBIOS Home page," Web reference: www.linuxbios.org
20. The XFree86 Project, Inc., "XFree86," www.xfree86.org
21. The GGI Project, "GGI: General Graphics Interface," Web reference: www.ggi-project.org
22. The DinX Project, "DinX is not X," Web reference: dinx.sourceforge.net
23. Gregory Haerr, "The Microwindows Project," Web reference: www.microwindows.org/MicrowindowsPaper.html
24. Trolltech Inc., "Qt, the Cross-Platform C++ GUI Toolkit," Web reference: www.trolltech.com
25. The SVGAlib Project, "Linux SuperVGA Graphics Library," Web reference: www.svgalib.org
26. Geert Uytterhoeven, "The Linux Frame Buffer Subsystem," Web reference: home.tvd.be/cr26864/Linux/fbdev
27. D. Stewart, "A Platform with Six Degrees of Freedom," Proceedings of the Institute of Mechanical Engineering, Vol. 180 (15), Part 1: 371-386, 1965-1966.
28. S. E. Landsberger, "Design and Construction of a Cable-Controlled Parallel Link Manipulator," Master's Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, September 1984.
29. S. E. Landsberger and T. B. Sheridan, "A New Design for Parallel Link Manipulators," Proceedings of the IEEE Conference on Systems Man and Cybernetics, pp. 812-814, Tucson, Arizona, November 1985.
30. Roger Bostelman, Nicholas Dagalakis and James Albus, "A Robotic Crane System Utilizing the Stewart Platform Configuration," Proceedings of the 1992 International Symposium on Robotics and Manufacturing, Sante Fe, New Mexico, November 10-12, 1992.



Copyright © 2002 by ASME